Programming HTML 5 Applications

根据HTML 5最新规范撰写，系统全面讲解如何利用
HTML 5和JavaScript开发强大的跨平台应用

HTML 5 应用开发实践指南

O'REILLY®
机械工业出版社
China Machine Press

Zachary Kessin 著
陈升想 汪奋进 译

O'Reilly精品图书系列

HTML 5应用开发实践指南

Programming HTML 5 Applications

[美]凯西恩（Kessin，Z.） 著

陈升想 汪奋进 译

ISBN：978-7-111-41451-3

本书纸版由机械工业出版社于2013年出版，电子版由华章分社（北京华章图文信息有限公司）全球范围内制作与发行。

客服热线：+ 86-10-68995265

客服信箱：service@bbbvip.com

官方网址：www.hzmedia.com.cn

新浪微博 @研发书局

腾讯微博 @yanfabook

目　录

O'Reilly Media,Inc.介绍

O'Reilly Media通过图书、杂志、在线服务、调查研究和会议等方式传播创新知识。自1978年开始，O'Reilly一直都是前沿发展的见证者和推动者。超级极客们正在开创着未来，而我们关注真正重要的技术趋势——通过放大那些“细微的信号”来刺激社会对新科技的应用。作为技术社区中活跃的参与者，O'Reilly的发展充满了对创新的倡导、创造和发扬光大。

O'Reilly为软件开发人员带来革命性的“动物书”；创建第一个商业网站（GNN）；组织了影响深远的开放源代码峰会，以至于开源软件运动以此命名；创立了Make杂志，从而成为DIY革命的主要先锋；公司一如既往地通过多种形式缔结信息与人的纽带。O'Reilly的会议和峰会集聚了众多超级极客和高瞻远瞩的商业领袖，共同描绘出开创新产业的革命性思想。作为技术人士获取信息的选择，O'Reilly现在还将先锋专家的知识传递给普通的计算机用户。无论是通过书籍出版，在线服务或者面授课程，每一项O'Reilly的产品都反映了公司不可动摇的理念——信息是激发创新的力量。

业界评论

“O'Reilly Radar博客有口皆碑。”

——Wired

“O'Reilly凭借一系列（真希望当初我也想到了）非凡想法建立了数百万美元的业务。”

——Business 2.0

“O'Reilly Conference是聚集关键思想领袖的绝对典范。”

——CRN

“一本O'Reilly的书就代表一个有用、有前途、需要学习的主题。”

——Irish Times

“Tim是位特立独行的商人，他不光放眼于最长远、最广阔的视野并且切实地按照Yogi Berra的建议去做了：‘如果你在路上遇到岔路口，走小路（岔路）。’回顾过去Tim似乎每一次都选择了小路，而且有几次都是一闪即逝的机会，尽管大路也不错。”

——Linux Journal

前言

这本书反映了Web技术的革命。之前，人们普遍认为Web无须编程，只要把脚本硬塞入网页之中就行了。现在，HTML和JavaScript在实现良好用户体验的过程中产生了重要作用。通过阅读本书，你将会掌握Web发展进程中最前沿的技术。

本书主要内容

本书覆盖的内容如下：

第1章　Web应用平台

介绍在新的HTML 5平台编程的原因，以及新平台为JavaScript编程人员提供的便利。

第2章　JavaScript的力量

介绍JavaScript中很强大但是你可能并不知道的功能，以及为什么你需要使用这些功能去探索本书涵盖的HTML 5的新特性以及相关库。

第3章　测试JavaScript应用

展示由JavaScript和浏览器提供的特定环境中测试程序的方法。

第4章　本地存储

描述使用localStorage和sessionStorage对象缓存在浏览器中的简单数据。

第5章　IndexedDB

展示支持本地存储的更强大的NoSQL数据库。

第6章　文件

描述从用户系统中读取和上传文件的方法。

第7章　离线处理

描述让用户在设备不能联网的情况下使用你开发的应用程序所要做的操作。

第8章　把工作分割成Web Worker

展示HTML 5和JavaScript的多线程能力。

第9章　Web Socket

演示如何通过使用Web Socket提高浏览器端和服务器端的数据传输效率。

第10章　新标记

总结HTML 5引入的对Web开发人员最为有用的新标记。

附录A　需要了解的JavaScript工具

描述本书中用到的以及其他可以让编程更快更准确的工具。

排版约定

本书采用了以下排版约定：

斜体（Italic）

表示新术语、网址、电子邮件地址、文件名和文件扩展名。

等宽字体（Constant width）

用于表示代码清单以及正文中引用的代码元素，如变量或函数名称、声明和关键词。

等宽斜体（Constant width italic）

表示需要由用户提供值或由上下文决定的值来替换的文本。

注意：表示提示、建议或者一般性注释。

警告：表示警告或注意事项。

代码示例的使用

本书旨在帮助你完成你的工作。一般来说，可以在程序和文档中使用本书的代码。如果你复制了代码的关键部分，那么你需要联系我们获得许可。例如，利用本书的几段代码编写程序是不需要许可的。售卖或出版O'Reilly书中示例的CD-ROM需要我们的许可。引用本书回答问题以及引用示例代码不需要我们的许可。将本书的大量示例代码用于你的产品文档中需要许可。

如果你在参考文献中提到我们，我们会非常感激，但并不强求。参考文献通常包括标题、作者、出版社和ISBN。例如："Programming HTML 5 Applications by Zachary Kessin（O'Reilly）.Copyright 2012 Zachary Kessin，978-1-4493-39908-5"。

如果你认为对代码示例的使用已经超出以上的许可范围，我们很欢迎你通过permissions@oreilly.com联系我们。

联系我们

有关本书的任何建议和疑问，可以通过下列方式与我们取得联系：

美国：

O'Reilly Media,Inc.

1005 Gravenstein Highway North

Sebastopol,CA 95472

中国：

北京市西城区西直门南大街2号成铭大厦C座807室（100035）

奥莱利技术咨询（北京）有限公司

我们会在本书的网页中列出勘误表、示例和其他信息。可以访问：

http://shop.oreilly.com/product/0636920015116.do

技术问题或评论本书，请发送邮件至：

bookquestions@oreilly.com

获取有关本书的更多信息、会议、资源中心和O'Reilly网站的相关信息，请查看我们的网站：

http://www.oreilly.com

http://www.oreilly.com.cn

在Facebook上联系我们：http://facebook.com/oreilly

在Twitter上联系我们：http://twitter.com/oreillymedia

在Youtube上联系我们：http://youtube.com/oreillymedia

致谢

每本书都是团队贡献的结果，如果没有一个强大的团队帮助我，我不可能完成这本书。首先，我必须感谢Simon St.Laurent给我写这本书的机会，并且在我创作的过程中不断地支持我。同样，我必须感谢Andy Oram，是他的编辑才能让这本书更优秀。还有，谢谢我的技术顾问Shelley Powers和Doinysios Synodinos，你们的反馈起了很大作用。

我还必须感谢Israeli开发者社区，比如我在Mytopia的前同事一年多来一直在项目过程中支持我，还有在Sayeret Lambda的伙伴们，现在Sayeret Lambda已经成为Tel Aviv交流编程心得的地方。

最后，谢谢我的妻子Devora一直以来对我的支持。没有她我不可能完成这本书。

第1章　Web应用平台

HTML 5可以将Web打造成创建真正应用程序的一流环境。HTML 5提供了对浏览器API的一系列关键扩展，以此加强了JavaScript现有的工具集，这些扩展使开发人员更简便地创建自我完善的应用，而非现在的这种仅仅显示远程服务器进程的界面。

Web最初是一种分享存储在Web服务器上文件的方法，这些文件只是偶尔变化。开发人员很快想出了如何实时生成那些文件，从而向构建应用程序迈出了第一大步。下一大步是向浏览器客户端中添加交互性。随着“浏览器大战”的肆虐和之后突然停止，JavaScript和文档对象模型（DOM）开始让开发人员能够创建动态HTML。几年后，Ajax将这些技术用于样式中。Ajax增加了一些工具，使页面与服务器以更小的单元进行通信。

HTML 5建立在这20年的发展之上，填补了一些重要的空白。从表面上看，许多HTML 5的变化是增加了对以前需要用插件实现的功能（尤其是多媒体和图像）的支持，实际上，它为JavaScript程序员提供了一些独立（或者至少更松散合作）创建应用程序的工具，有了这些工具他们可以用HTML搭建架构、用CSS制作界面、用JavaScript表达逻辑和行为。

为Web应用增加力量

HTML 5提高了Web应用的标准。尽管它仍然需要工作在安全约束条件下，但最终会提供桌面开发人员已经期盼多年的工具：

本地存储

引用键-值系统多达5MB的数据。

数据库

起初是一个基于SQLite的API，形势似乎已经转向IndexedDB,IndexedDB是一种JavaScript原生的NoSQL系统。

文件

尽管出于安全的原因Web应用依旧不能自由地访问文件系统，但是已经能够使用用户指定的文件，并开始创建文件了。

离线操作

当笔记本电脑或手机在“飞行模式”时，We b应用无法与服务器通信。清单（Manifest）文件通过缓存文件到本地帮助开发人员实现离线应用。

Web Worker

线程和子进程一直是个问题，但是JavaScript根本不提供这些。Web Worker提供了一种方法，将应用进程放到独立空间中，在那里它们可以互不干扰地工作。

Web Socket

尽管超文本传输协议（HTTP）随着时间的推移有一些更新，但它一直是Web的基础。Web Socket将“请求/响应”的通信方式转化成创建更灵活的通信系统。

当然，还有更多工具，从地理信息、音频和视频、画布图形到各种小的新标记，这些工具为在HTML 5上构建工业级的应用提供了基础。

开发网络应用程序

过去，复杂的Web应用程序只不过是一个目录、一个从数据库派生的静态页面或者一个JavaScript生成的计算器，没有人梦想用JavaScript做复杂的应用程序。复杂的应用程序需要用Java、C或C++写的专门的客户端/服务器应用程序。事实上，在DOM和Ajax出现前想要用JavaScript这样做几乎是不可能的。然而，Ajax引入了不需要重新加载页面即可与服务器通信的功能，而且DOM允许程序员即时改变HTML。

2007年Google引入了Gears，这是一个浏览器扩展，它带给开发人员前所未有的力量。Gears允许浏览器离线工作，可以让用户在浏览器中存储更多数据，并设计了一个工作池（Worker pool）用于处理长时间运行的任务。Gears已经停用，但是它的大多数功能经过修改后已移植到HTML 5中。

现代Web看起来像一个全方位的网站，包含各种内容，从维基百科那样依旧有效的旧式文档集合，到Facebook、YouTube和eBay这种提供与他人交互的网站，再到可以称为替代桌面应用程序的东西（例如Gmail和Google Docs）。许多以前独立的应用程序，例如邮件客户端，已经变成了Web体验中不可分割的一部分。在现代Web中，应用程序和页面间的界线已经非常模糊，只在网站的用途上有所区别。

在浏览器中运行应用程序对于用户和开发人员都有很大的优势。对用户来说，使用Web应用没有负担：用户可以试用应用程序，如果不喜欢就换个页面而不会在硬盘上留下任何东西。尝试新的应用也相当安全，因为它们运行在一个沙箱环境中。当开发人员更新代码时，新版本的应用程序会自动下载到浏览器。Web应用程序很少有版本号，至少已公开的应用程序是这样的。

对开发人员来说，在浏览器中运行应用程序优势更大。首先，对用户来说的优势对开发人员也是优势。其次，开发人员不需要写安装程序，而且新版本可以自动发送给用户，使得小量的更新切实可行。另外，还有其他好处。

Web是跨平台的，开发人员都希望开发的应用能运行在多个系统上，例如Windows XP、Windows Vista、Windows 7、Mac OS X、Linux、iPhone/iPad和Android，这个愿景如果用传统的开发工具来实现将是一项非常艰巨的任务，但是用Web和一些具有前瞻性的技术来实现，就会变得十分简单。一个用标准类库（例如jQuery）建立的网络应用能运行在上述所有这些和其他几个平台的多数浏览器上。Sun公司一度希望Java applets将Web定义为一个平台，现在JavaScript已成为默认的Web平台。

你甚至可以在移动设备上运行Web应用，至少现在可以在智能手机上运行。你可以用一个包装（例如PhoneGap）来创建一个HTML 5

应用，并把它打包，在App Store、Android Market和其他网站上出售。你可以创建一个与Web服务器大量交互或者完全独立的应用，两者皆可。

HTML 5之前的Web真正的不足之处在于，一个运行在计算机上的Web应用要占用上千兆字节内存和磁盘空间，运行起来像在老式vt320终端上一样慢。所有数据存储必须在服务器上完成，所有文件必须从服务器加载，每一个交互必须完成一次与服务器间的往返。这会使用户感觉较慢，特别是当用户距离服务器很远时。如果用户查看页面每次要耐心等待至少400ms，用户就会感觉应用程序运行很慢。从我在特拉维夫的办公室到加利福尼亚的一个服务器，ICMP ping的一次往返时间约为250ms。到服务器上的任何操作都将耗费额外的时间，使应用运行慢下来。当然，移动设备通信更慢。

JavaScript的胜利

尽管JavaScript自1995年出现以来已经成为Web开发的关键组成部分，但是这十几年来它一直背负着坏名声：语法古怪、性能低下、产生奇怪的错误，以及对DOM（Document Object Model，文档对象模型）的依赖。尽管浏览器已经将其封闭在“沙箱”中，简化了用户的安全问题，但是要为JavaScript开发人员提供一些在传统桌面应用开发中显得微不足道的功能却很难。

脚本文化造就了它自身的问题。提供非常低的进入门槛是件好事，但是需要付出代价。代价之一就是，这样一种语言往往允许没有经验的程序员做一些非常不明智的事情。起初，程序员可以很容易地从Web上找到JavaScript例子，剪切、粘贴或做一些修改，就能得到大多数可用的东西。遗憾的是，随着时间的推移对这些代码的维护变得越来越困难。

随着Ajax的流行，开发人员对JavaScript有了新的看法。一些开发者致力于对解释和运行JavaScript代码的引擎进行改进，致使运行速度大幅度提升。还有一些开发者则致力于语言本身，他们意识到JavaScript有一些很不错的功能，能够开发最好的应用，正如Douglas Crockford的《JavaScript语言精粹》（JavaScript：The Good Parts,O'Reilly 2008出版）中所讲述的一样。

除了核心语言之外，开发人员还创建了一些工具使JavaScript更易于调试。尽管Venkman（一款早期的调试器）在1998年就出现了，但是2006年发布的Firebug成了JavaScript调试器的黄金标准。它允许开发人员对Ajax调用进行跟踪，查看DOM和CSS的状态，并单步执行代码等。基于WebKit的浏览器提供了类似的内置功能，尤其是苹果公司的Safari和GoogleChrome,Opera Dragonfly则对Opera提供支持。甚至，在移动设备封闭空间中工作的开发人员现在也能用Weinre（Web Inspector Remote：远程Web调试器）像Firebug一样进行调试。

类库是JavaScript最近大规模投入的最终关键组成部分。开发人员可能依旧不能理解他们使用的全部代码，但是能够将代码组织得更易于升级，有时甚至可以用可互换的类库简化代码管理。

jQuery

如果有一个类库可以称为JavaScript类库的“黄金标准”的话，那一定是John Resig的jQuery类库。jQuery围绕DOM和其他JavaScript对象（如XMLHttpRequest对象）形成了一个包装器，使得用JavaScript实现各种功能更轻松、更有趣。从许多方面讲，jQuery都是每个JavaScript程序员应该熟知、必不可少的类库。学习jQuery请访问网站http://jquery.org，或者阅读由O'Reilly出版的大量优秀图书，例如《深入浅出jQuery》（Head First jQuery）或者《jQuery锦囊妙计》（jQuery Cookbook）。本书中的许多示例都是用jQuery编写的。

ExtJS

鉴于jQuery形成了一个DOM包装器ExtJS，因此人们从Sencha（http://sencha.com）开始设法尽可能地把它从中剥离。ExtJS以丰富的widget控件集为特点，可以在任意Web页面上执行，并提供了许多桌面开发人员所熟悉的widget控件，如树、网格、标单、按钮等。整个系统经过深思熟虑并良好地组织在一起，使许多应用的开发成为一种享受。虽然ExtJS的库往往都很大，但是它非常适合一些应用开发。ExtJS有一个不错的功能，即它的许多对象都知道如何保存自身的状态。因此，如果用户有一个表格，并当再次浏览该表格时对它的列进行了重新整理，那么可以设置它来保存这个状态。第4章中的“在ExtJS中使用localStorage”讲述了如何利用本地存储加强这一功能。

Google Web套件及其他

一些工具如GWT允许程序员编写Java代码，然后编译成JavaScript，这样就可以在浏览器中运行了。

第2章　JavaScript的力量

JavaScript语言编程并不难，但是要达到真正的JavaScript专家级水平非常有挑战性。成为熟练的JavaScript程序员有几个关键因素。本章中的技术将反复出现在本书剩余部分所介绍的类库和编程实践中，因此在继续介绍其他章节前需要先熟悉这些技术。

附录中列出了一些优秀的JavaScript编程工具。这些工具可以提供大量的帮助。例如，JSLint（见附录：JSLint）能捕获一些程序员可能会漏掉的错误。一些网站是这类编程工具的很好来源，如StackOverflow和O'Reilly Answers。

本章并非要完整地介绍JavaScript的力量。有关内容可参阅下列JavaScript书籍：

·《JavaScript语言精粹》（JavaScript,The Good Parts），Douglas Crockford

·《JavaScript权威指南》（JavaScript：The Definitive Guide），David Flanagan

·《高性能JavaScript编程》（High Performance JavaScript），Nicholas C.Zakas

·《JavaScript模式》（JavaScript Patterns），Stoyan Stefanov

非阻塞I/O和回调

撇开语言本身不谈，学习JavaScript的首要关键是理解事件驱动编程。JavaScript运行环境中的操作往往是异步操作，这就是说先在某个地方创建操作，当一些外部事件发生后再执行。

这样可以说明传统语言中I/O方面发生的主要变化。例2-1是一个典型的传统语言I/O示例，本例使用PHP语言。$db-＞getAll（$query）；这一行要求数据库访问硬盘，因此它花费的运行时间将比函数中的其他部分多很多。当程序等待服务器执行时，查询语句被阻塞，程序什么都不做。通常，这在服务器端语言中（如PHP）不成问题，因为可以有很多并行执行的线程或进程。

例2-1：PHP中的阻塞I/O

```
function getFromDatabase()
{
$db=getDatabase();
$query="SELECT name FROM countries";
$result=$db-＞getAll($query);
return$result;
}
```

然而，JavaScript中仅有一个执行线程，因此，如果函数被阻塞就什么都不做了，那么用户界面也会冻结。因此，JavaScript必须找到一

个不同的方式来处理I/O（包括所有的网络操作）。JavaScript所做的就是立即从一个可能感觉缓慢的方法中返回，留下一个函数，当操作（例如，从Web服务器下载新的数据）完成时进行调用。该函数称为回调。当Ajax调用服务器时，JavaScript发出该请求，然后继续做别的事情。它还提供了一个函数，在服务器调用完成后调用它。这个函数通过服务器返回的数据被调用（回调因此而得名），这时数据已经准备就绪。

打个比方，考虑在杂货店买东西的两种方式。一种方式是，一些商店把东西放到柜台后面，你必须询问售货员，并等待她取来你想要的东西，这就像刚才所示的PHP程序。另一种方式是，一些商店有一个熟食柜台，你可以订购并得到一个订单号。你可以离开去买别的东西，等订单准备就绪时取走。这种情况就像一个回调。

通常，一个快速的操作可以是阻塞式的，因为需要马上返回其所请求的数据。一个缓慢的操作，例如调用服务器可能需要花费几秒钟的时间，应该是无阻塞的，并通过一个回调函数返回其数据。函数中存在回调函数选项将为计算运行一个操作所需的相对时间提供良好的线索。在单线程语言如JavaScript中，函数阻塞着等待网络或用户时，浏览器不可能不锁定。

因此，策略地使用回调是掌握JavaScript重要的一步，知道它们什么时候触发。例如，当使用Ajax的数据存储对象时，数据一两秒后就

没有了。使用一个闭包（见本章后面的“闭包”）来创建回调函数是处理数据加载的正确方法。在JavaScript中所有这些外部的I/O（数据库、调用服务器）都应该是非阻塞的，因此学习使用闭包和回调至关重要。

在一些应该避免的例外情况下，JavaScript的I/O不会阻塞。三个主要的例外是窗口方法alert()、confirm()和prompt()。从这三种方法调用开始到用户关闭对话框的那一刻，它们阻塞了页面上的所有JavaScript。此外XHR对象可以使Ajax以同步模式调用服务器，此方式在Web Worker中可放心使用，但是在主窗口中它会引起浏览器UI锁定，应避免使用。

强大的Lambda函数

从PHP或其他程序语言转向JavaScript开发的程序员，往往会像在他们用过的语言中那样处理JavaScript函数。虽然这样使用JavaScript函数也可以，但是却错过了很多使得JavaScript函数变得强大的东西。

JavaScript函数可以用函数语句（见例2-2）或函数表达式（见例2-3）创建。这两种形式看起来非常相似，都产生一个名为square的函数，该函数可以计算数的平方。然而也有一些关键的差别。第一种形式是加载方式，这就是说，该函数将在闭包的开始被创建。所以想有条件地定义函数时，不能使用函数语句，因为JavaScript不会等待条件语句执行后再决定是否要创建函数。在实践中，大多数浏览器允许把函数放在if里面，但这不是一个好主意，因为在这种情况下浏览器做的事情可能会有所不同。如果一个函数的定义应该是有条件的，那么用函数表达式更好。[1]

例2-2：函数语句

```
function square(x){
return x*x;
}//注意：少一个"；"
例2-3：函数表达式
var square=function(x){
return x*x;
};
```

在第二种形式——函数表达式中，函数在程序的流程执行到这一点时创建。可以有条件地定义一个函数，或在一个较大的语句内部定义函数。

此外，函数表达式没有给函数命名，因此函数可以为匿名。然而，这个例子在等号的左侧指定了一个函数名（square），这是个好主意，原因有两个。首先，当调试程序时，指定一个函数名，让你可以辨别堆栈跟踪过程中看到的是哪个函数，没有它，该函数将显示为anonymous。如果用Firebug跟踪堆栈时看到一个堆栈中有九个或十个显示为anonymo us的函数将令人非常沮丧。其次，指派一个函数的名称，还能够根据需要递归调用的函数。

JavaScript中的函数表达式可以在任何能用表达式的地方使用。因此，可以像例2-3一样把函数分配给一个变量，也可以分配给一个对象成员或传递给函数。

JavaScript函数比C函数更像Lisp中的Lambda表达式。在C类型语言中（包括Java和C++），函数基本上是一个静态的东西。它不是一个可以操作的对象。虽然可以把一个对象作为参数传递给函数，但是几乎没有能力来构建复合对象，或其他扩展对象。

早在20世纪50年代，当时Lisp刚刚被创建，麻省理工学院的人深受Alonzo教会Lambda Caculus（λ演算）的影响，λ演算提供了一个处

理函数和递归的数学框架。因此，约翰·麦卡锡用关键字Lambda来处理匿名函数。这种做法已传播到其他语言中，例如Perl、Python和Ruby。虽然关键字"Lambda"没有出现在JavaScript中，但是其函数做着同样的事情。

JavaScript中的函数如同Lisp语言中的函数一样是“一等公民”。JavaScript中的函数仅仅是一个具有特殊属性可以被执行的数据。函数可以像JavaScript的所有其他变量一样操作。在C和类似的语言中，函数和数据作用于两个独立的空间。而在JavaScript中，函数就是数据，并且可以用在每一个可以使用数据的地方。函数可以分配给一个变量、作为参数传递或作为函数的返回值。在JavaScript中，将一个函数传递到另一个函数是很常见的操作。例如，可以用于为一个按钮点击创建回调函数（见例2-4）。另外，还可以通过简单的赋值改变函数。

例2-4：ExtJS用函数作为处理函数的按钮

```
var button=new Ext.Button({
text：'Save'，
handler：function(){
//这里进行保存
}
})；
```

[1] 这个地方原文中是“函数语句”，应该是作者写错了，因为上面说了“当你想有条件地定义函数时，不能使用函数语句”。

闭包

如果没有闭包（Closure）带来的好处，那么在JavaScript中访问作为第一类对象的函数就不太有价值了。闭包是从Lisp语言移植到JavaScript中的又一个元素。当一个函数在JavaScript中创建时，该函数可以对其生成环境中任何语法空间的变量进行访问。即使最初定义它们的上下文已执行完毕，但是这些变量仍然可用。该变量可以被内部函数和外部函数访问、修改。

闭包对构建回调是有用的。函数任意时刻都可能运行来响应一些事件，但是需要知道之前发生了什么。

这在构建函数生成器时非常有用，因为每当生成器函数运行时将有一个不同的外部状态，闭包在创建函数中。也可以在一个生成器中创建多个函数，对于同一个环境这些函数都是关闭的。

闭包是JavaScript函数最强大的特征之一。在简单的情况下，它可用于创建访问外层空间变量的函数，从而允许回调访问控制函数中的数据。然而更强大的是能够创建把变量绑定到一个范围内的自定义函数。

如例2-5所示，DOM元素或CSS选择器"el"被包装在一个函数中，通过一个简单的函数调用对HTML内容进行设置。外层函数

（factory）将"el"元素绑定到内层函数所使用的一个变量，通过jQuery设置DOM元素。外层函数将内层函数作为返回值返回。这个例子的结果是把变量updateElement的值设置为内层set函数，并且参数el的值是一个已绑定的CSS选择器。当一个程序调用update Element并且传入CSS选择器后，updateElement会返回一个可用于设置与该HTML相关的HTML元素的函数（即用于设置该CSS选择器选中的HTML元素的函数）。

例2-5：简单闭包

```
var factory=function factory(el)
{
return function set(html)
{
$(el).html(html);
};
};
```

也可以创建多个在同一空间内关闭的函数。如果一个函数把多个函数返回到一个对象或数组中，所有这些函数都有机会获得创建函数的内部变量。

例2-6所示代码向浏览器的工具栏添加了一个tools数组中定义的按钮。每个按钮都有其自己的处理程序（命名为clickHandler）。该函数访问调用函数的变量，并将button和tool变量嵌入到它的操作中。您只

需要轻松地在tools数组中添加或减去一个元素更新应用程序，定义了所有功能的按钮就会出现或消失。

例2-6：按钮闭包

```
$('document').ready(function Ready(){
var button,tools;
tools=['save', 'add', 'delete'];
console.info($('div#toolbar'));
tools.forEach(function(tool){
console.info(tool);
var button=$('＜button＞').text(tool).attr({
css: 'tool'
}).appendTo('div#toolbar');
button.click(function clickHandler(){
console.info(tool,button);
alert("User clicked"+tool);
});
});
});
```

使用闭包时，很难知道哪些变量在或不在一个函数的空间内。然而，无论Google Chrome的开发工具还是Firebug都能显示已闭包的变量列表。

在Firebug中，可以在Script选项卡的"Watch"中找到空间链。在当前空间下的所有变量是一个上升到主要"window"对象的空间层次。

例如，在Google Chrome的开发工具中，当代码在断点停止时，右侧栏Scope Variables标记的"Closure"标记将显示当前函数（见图2-1）

的内部变量。在这种情况下，它显示我们点击了"delete"按钮，并列出了按钮本身引用的jQuery对象。

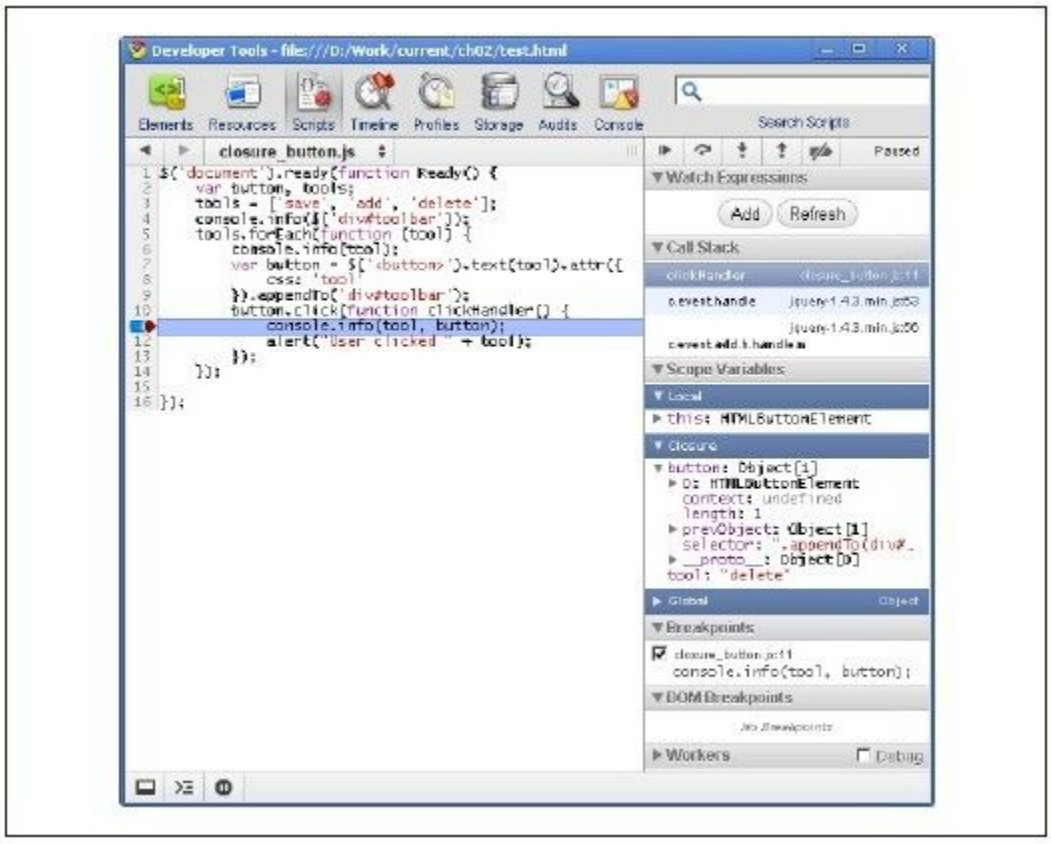

图　2-1　Google Chrome开发工具中的闭包

函数式编程

函数式编程是一种与Lisp、Scala、Erlang、F#或者Haskall语言相关的方法，在JavaScript中也相当好用。函数式编程依赖于一些基本假设：

函数是该语言的“一等公民”，并可以在任何能使用其他值的地方使用它。

可以通过组合简单的函数构建复杂的行为。

函数有返回值，在多数情况下，对于同样的输入给定函数总是返回相同的值。

在数学中，函数没有“副作用”，如经典数学函数y=sin（x）。它只返回一个y可以存储的值，而不会改变x或程序中任何全局状态的东西。确保函数是“纯的”（没有“副作用”），这种做法使函数能在程序中的任何地方调用，而不会发成一些奇怪的事情。在编程时，副作用所带来的问题是，可能会导致奇怪的、非常难以追踪的依赖关系。如果调用的方法可能会导致数据在其他地方破坏，就会使潜在的错误大大增加，那将是非常难以追踪的。

JavaScript函数可能有“副作用”，而且也没有内置的方法防止函数产生副作用。此外，JavaScript函数默认不返回值，除非已经显式调用了一个return语句来返回一个值。在没有返回语句时函数将返回undefined。

当应用函数式编程时，程序员往往形成一种工作模式，使用许多非常小的函数，每个函数往往只用两三行代码来完成一个目标。这是一种很好的设计技术，因为通常很短的函数更容易保证正确性，测试也更容易。

常见的情况是，可以通过组合简单函数生成复杂的行为。可以用简单的函数生成一个函数链，使链中的每个函数都返回this，从而允许调用下一个函数。

jQuery类库经常使用例2-7所示的函数链。在这个例子中，jQuery找到一个DOM项，设置其文本，淡化到视图，然后给它设置一个单击处理函数，用第二个DOM链隐藏它。

例2-7：用闭包生成函数链

```
$('div.alert').text("Message").fadein
(2000).click(
function()
{
$(this).fadeout(2000);
}
);
```

函数式编程的一个非常强大的模型是高阶函数。高阶函数用一个函数作为参数抽象出特定的行为，而将通用行为留在外层函数中。

高阶函数的一个很好的例子是数组映射函数。本章后面的“数组迭代操作”取一个数组并返回一个新的数组，新的数组是被传递的函数应用到数组的每一个元素得到的结果。该模型的应用范围很广泛，不仅仅是数组操作。作为通用模型，高阶函数可用于需要具体修改通用行为的任何地方。

jQuery库的接口往往有利于函数式编程。jQuery接口特意优化了从页面中选择一组DOM节点的方法，然后为这些节点的交互提供一个函数接口。

此外多数jQuery方法返回一个值，使它们能够被链接。例如，要在页面中找到比设置尺寸宽的所有的图片，可以选择页面中的所有图片，过滤掉那些小于300像素的，然后按比例缩放列表中剩下的图片。

例2-8正是这么做的。它选择文档中的所有图片（任何含有img标记的文件），然后过滤那些宽度大于300像素的（maxWidth），并按比例缩放。通过简化过滤器和缩放函数，可以确信代码将按我们的意图工作。

例2-8：缩放图片

```
var scaleImages=(function(maxWidth)
{
return function()
{
$('img').filter(function()
{
return$(this).width()＞maxWidth;
}).each(function()
{
$(this).width(maxWidth);
});
};
}(300));
```

当在一个耗时的程序中处理操作列表时，例如Ajax调用，有时用一个请求发送整个列表到服务器是不现实的。例如，发送整个列表可能会导致服务器超时。

在这种情况下遍历列表，将列表看做一个头部和尾部将非常有用。取列表的第一个元素（或前几个元素）进行处理，然后通过使用递归处理列表的其余部分，直到列表为空（见例2-9）。

我已经用这个方法将数据添加到一个REST界面。每调用界面一次平均需要大约1s，所以在Ajax调用中调用它500次很不实际。在这种情况下，可以通过递归处理列表。

注意：术语Ajax和XHR究竟是什么？JavaScript对象被称为XMLHttpRequest，缩写为XHR。Ajax来自"Asynchronous JavaScript and XML"（异步JavaScript和XML），由Jesse James Garret创建。事实

上，在许多情况下，通过网络发送的数据不是XML，而可能是JSON或其他数据格式。

例2-9：列表递归

```
/*列表递归示例*/
function iterateAjax(tasks){
function iterator(tasks){
$.ajax({
url：'index.php'，
data：tasks[0]，
success：function(data,status,XMLHttpRequest){
if(tasks.length＞0){
//用这里的结果做点事儿
iterator(tasks.slice(1))；
}
}
})；
}
iterator(tasks)；
}
```

虽然只用函数式编程风格建立一个完整的单页Web应用程序是不实际的，但是不应该忽视其中许多有用的想法。例如，函数式编程对使用Web Worker非常适合（见第8章）。

JavaScript中关于函数式编程没有介绍太多，但相当多的其他语言可以应用到JavaScript中。了解函数式编程的更多信息，请参阅下列书籍：

·《Real World Haskell》，作者Bryan O'Sullivan、John Goerzen、Donald Bruce Stewart（O'Reilly）

·《Programming Scala》，作者Dean Wampler、Alex Payne（O'Reilly）

·《Structure and Interpretation of Computer Programs》，作者Harold Abelson、Gerald Jay Sussman（MIT Press）

原型及如何扩展对象

JavaScript中的一切都可以有附加的方法。每一个元素都有一些供程序员使用以提高有效性的基本方法。JavaScript基元如布尔、字符串和数字作为对象都有第二次生命。从基元到对象的转换是透明的，所以可以在基元上应用这些方法。实际上，发生的事情是将一个简单的值（例如一个字符串）转换为一个对象，如果需要的话再转换回来。

对于字符串，可以调用大量的方法来操作。一些方法能按位修改字符串，大部分将返回一个新的字符串。在Mozilla开发者网站（http://developer.mozilla.org/en/javascript）上可以找到一个完整的列表，这里仅列出最重要的几个：

string.indexOf()

返回字符串中第一个与给定子串匹配的子串序号，当没找到时则返回-1。

string.lastIndexOf()

与indexOf()相同，不过是从结尾开始。

string.match()

在一个字符串匹配一个正则表达式。

string.replace()

替换正则表达式（指定为函数或字符串）。

string.split()

把一个字符串分割成一个子串数组。

string.slice()

生成一个子串。

提取一个子串

然而，有时会遇到预定义方法不够，又需要一些自定义功能的情况。在这种情况下，JavaScript提供了一种不常用但很强大的特性：一种扩展内置对象的方法。虽然你总是可以用一个简单赋值给JavaScript对象分配一个方法，但是这不一定是最好的方式。如果想给所有字符串添加一个方法，可以把该方法附加给String.prototype对象。在JavaScript对象系统中，每个对象都继承自原型链，因此通过在链中的某个地方增加方法，可以添加到整个这类对象。

下面用一个例子来说明这个概念。目标是创建一个名为populate的新方法，把值用一个模板替换。模板就是调用该方法的对象，例如：

```
Hello{name}
```

该字符串应该在大括号中包含程序员想用指定值替换的关键字。Populate的参数是一个对象，在模板中指定关键字和替代值。因此，如果参数包含一个名为name的属性，则name的值就会插入字符串中。

例2-10的代码运行后，populate方法就会附加到所有字符串中。当populate被调用时，将引用标准的JavaScript对象this调用的字符串。有了this的值，populate方法可以利用简单的替换在它的参数中插入值。

在一般情况下，最好不修改调用了方法的对象，但是要返回一个新的对象实例（函数式编程想法）。

例2-10：替换字符串标识

```
   String.prototype.populate=function populate(params){
   var str=this.replace(/\{\w+\}/g,function stringFormatInner(word)
{
   return params[word.substr(1，word.length-2)]；
   })；
   return str；
   }；
   $('.target').html("Hello{name}".populate({
   name："Zach"
   }))；
```

当然，字符串不是JavaScript中唯一具有原型的对象类型，数字、数组、对象、函数和布尔做的一样好。

警告：扩展基本对象的原型，例如Object、Array等，有时会破坏类库。罪魁祸首通常是要创建的属性已经存在于对象中。要先确认正在创建的属性不存在，然后再添加属性，并仔细测试。

事实上，在JavaScript中扩展基本类型是一种有很大争议的实践。有人说，不应该这么做。我认为这是一个强大的工具，不能全盘否定。

Number原型的工作方式与字符串原型完全相同。因此，可以定义一个新的方法，处理可能提出的任何需求。例如，如果一个应用程序

需要定期请求计算平方数，可以非常容易地添加该方法（见例2-11）：

例2-11：使用Number.prototype

```
Number.prototype.square=function square(){
return this*this;
};
6.square(); //36
```

用原型扩展函数

除了数据对象如字符串和数组外，函数也有原型，可以用来创建非常强大的复合函数。通过把简单函数组合成较大的单元，使用Function.prototype给Function对象添加方法，可以把复杂的逻辑分割成许多简单的情况。事实上，许多工具包正是这样工作的，并提供了一些方法来完成这些任务。

有一个原型的实例，可以通过执行它提高代码的健壮性，那就是在函数执行之前添加错误检查。在例2-12中，cube函数运行时不检查其输入是否为数字。代码中函数包装在一个负责检查的拦截器中。每当cube在后面被调用时，拦截器先运行，如果输入的是一个数字再调用cube。

例2-12：函数拦截器

```
   Function.prototype.createInterceptor=function
createInterceptor(fn){
   var scope={};
   return function(){
   if(fn.apply(scope,arguments)){
   return this.apply(scope,arguments);
   }
   else{
   return null;
   }
   };
   };
   var interceptMe=function cube(x){
```

```
console.info(x);
return Math.pow(x, 3);
};
var cube=interceptMe.createInterceptor(function(x){
return typeof x==="number";
});
```

举一个更广泛的例子，你要用常规方法计算斐波那契（Fibonacci）数列。一个非常简单暴力的方法类似于例2-13。然而，fib（40）这种琐碎函数运行起来需要相当长的时间。

例2-13：简单斐波那契数列计算方法

```
var fib=function fib(n){
if(n===1||n===2){
return 1;
}
return(fib(n-1)+fib(n-2));
};
```

通过对该函数示例的运行进行快速跟踪，可以看到它做了大量冗余计算。如果让Fibonacci方法对每个值只作一次计算，运行将快很多。我们可以通过用拦截器的方法包装Fibonacci函数，缓存每次迭代的结果（见例2-14）。拦截器不需要知道Fibonacci数列是如何产生的，只需要知道对于一个给定的输入，应该总是产生相同的输出。所以一旦fib（n）被计算，查询就变得非常简单，如果没有查询到结果，可以通过计算得到。

例2-14由两部分组成：缓存方法和实际的Fibonacci数列生成器。缓存方法不知道任何关于Fibonacci数列的事情，除了一个事实：一个给定的输入值将总是返回相同的值，这可以被缓存。因此，当decoratedFib（32）被调用时，缓存会先检查是否已经为32计算出了结果。如果有结果，它只需要直接返回。如果还没有结束，它会开始计算。Fibonacci数列的计算是非常复杂的递归，因此计算32的Fibonacci数列必须先计算31、30等。该函数将递归地寻找结果，直到找到为止。如果这是函数第一次运行，将在种子值中找到n=2和n=1。

虽然很多人不会在Fibonacci数列上花费太多时间，但是它是一个很好的使用函数原型的例子，可以用两个很短的函数结合得到非常强大的结果。

注意：这些例子复杂多了，其实是为了显示如何从函数缓存没有副作用的函数结果。实际上这可能不是写这个代码最好的方式。

例2-14：高级斐波那契数列计算方法

```
var smartFib=(function makeFib(){
var fibsequence=[0，1，1]；
var fib=function fib(n){
if(fibsequence[n]){
return fibsequence[n]；
}
fibN=fib(n-1)+fib(n-2)；
fibsequence[n]=fibN；
return fibsequence[n]；
}；
return fib；
```

```
}());
Function.prototype.decorate=function Decorate(params){
return params.decorator(this,params.initialData);
};
var cache=function cache(lambda,initial){
return function cacheRunner(n){
if(initial[n]!==undefined){
return initial[n];
}
else{
initial[n]=lambda(n);
return initial[n];
}
};
};
var decoratedFib=function fib(n){
return decoratedFib(n-1)+decoratedFib(n-2);
}.decorate({
decorator: cache,
initialData: [0, 1, 1]
});
```

假设一个函数定期运行响应用户输入，但是在给定的时间内运行次数不能超过一次。这很简单，当该函数最后被调用时，用函数原型创建一个要存储的包装函数，让它在指定时间内不能再次运行。可以根据应用程序的需要选择不运行它，或者抛出一个异常。

在另一面，也可以给函数创建一个方法，在一个延时后执行或定期执行。通常，一个任务需要在一个事件后运行，但不应该运行过于频繁。例如，你可能要检查用户的输入，但是每次按键后都运行检查有些过分。设置一个每250ms运行一次的方法可以解决这个问题。

这有两种实现方式。第一，可以运行一次该方法，并且在到达指定时间前不允许它再次运行。第二，可以创建一个方法，使它在调用

后会以一定时间间隔运行，并且在调用时复位定时器。如果我们的目的是当用户打字停止或一些其他的事件暂停时运行某些代码，这种模式将非常有用。在实践中，一个新的方法将作为包装了基本JavaScript的setTimeout()和setInterval()方法的包装器运行，使用更加方便。也可以创建一个方法根据预设安排今后的任务或取消现有的任务。

柯里化和对象参数

在函数式编程中，一个常见的编程模型是“柯里化”（currying）一个函数。因Haskell语言的一个特点而得名，柯里化是指将几个参数组合成一个单一对象的做法，这样就可以把它们作为一个参数传递给函数。如果一个函数需要大量的参数，在JavaScript中最好是放弃长长的参数列表，并用一个单一对象作为参数。通过将一个对象作为参数，可以把各种选项全部转换成名称/值对。这样做的好处之一是，参数的顺序变得无关紧要。此外，可以创建一些或全部的参数选项。对于接收了许多选项的复杂方法会有很大帮助。特别对一些ExtJS中的对象创建方法往往非常有用。

柯里化参数的最简单方法是创建一个函数，用它接收参数块，并返回一个函数，返回的函数将以预先提供的默认参数调用原函数（见例2-15）。这样就可以建立一套默认值，不必每次都指定，同时允许调用者改变他们想要修改的任何参数。

例2-15：柯里化示例

```
Function.prototype.curry=function FunctionCurry(defaults){
var fn=this;
return function(params){
return fn.apply(this,defaults.concat(params));
};
};
```

这种模式也可以应用于创建对象。作为一个接受参数块的对象构造函数，可以用一个自定义类作为对象的子类，调用父类的构造函数，调用时子类将用一套默认参数覆盖用户传递的参数。

数组迭代操作

像其他JavaScript中的第一类对象一样，数组也有方法。标准数组有一些为程序员设计的方法。在较新版本的Firefox浏览器中（1.5以上版本），也已创建了一些标准的迭代方法。

这些操作的基本思路是：处理一个lambda函数，并将其应用到数组中的每个元素，得到一些结果。通过几个小函数使用这种方法，可以处理一个数组，并创建它的操作集，该操作集可以建立一个代数数组。这反过来又可以让你用基本的操作集合建立非常强大的操作集。

第一个数组方法是map()。map函数包含一个数组和一个方法作为参数。它把该方法应用到数组的每一个元素，并用返回值创建一个新数组。所以给定一个数字的数组，通过在map()中简单地对每个数字应用square函数，可以创建一个平方数的数组。

注意：数组方法如map在多数现代浏览器上都有（它们已经被加入IE 9）。然而，如果没有也可以添加该功能。在Mozilla的开发者网站上可以找到所有数组方法的示例代码。

调用的函数接收数组的当前值、数组中当前位置的序号和整个数组的序号作为参数。通常情况下，map工作者函数只需要查看数组的当前值，见例2-16。

例2-16：数组Map

```
[1，2，3，4，5].map(function(x){
return x*x;
});
//结果：[1，4，9，16，25]
```

不过，在一些情况下这可能还不够。例2-17的目的是显示数组中一个值的平均运行次数，因此每个元素需要了解其相邻元素。

例2-17：平均运行次数

```
function makeRunningAverage(list,size)
{
return list.map(function(current,index,list)
{
var start,end,win;
/*找到滚动平均值窗口的开始点和结束点*/
start=index-size＜0?
0：
index-size;
/*提取该窗口*/
end=index+size＞list.length?
list.length：
index+size;
win=list.slice(start,end)；/*取平均值*/
return win.reduce(function(accumulator,current)
{
return accumulator+current;
}，0)/(end-start);
});
}
```

使用for循环操作数组元素有几大优势。首先，它逻辑清晰并与迭代代码隔离，允许程序员将数组视为一个整体。其次，在避免副作用

方面，作为内部函数并保持函数简短，可以写出非常健壮的代码。

另一个应考虑的情况是增加回调处理函数。用一个for循环遍历元素并增加处理函数，可以不使用JavaScript函数的封闭属性。你可能会使用类似于例2-18所示的方法，但是这样达不到预期目的。在这种情况下，该方法将总是显示最后一个元素。在这里具有欺骗性的是，在任何情况下引用的i值都是最后一个值。闭包总是能知道变量的当前值，而不是它创建时的值。当for循环遍历列表时，会改变i的值。在例2-19中，代码将正常工作。在这种情况下，每次迭代所引用的节点都是独立的，它处于函数空间的内部。

例2-18：for循环

```
for(var i=0；i＜nodes.length；i+=1){
nodes[i].bind('click'，function(){
console.info(nodes[i])；
})；
}
```

例2-19：绑定forEach

```
nodes.forEach(function(node){
node.bind('click'，function(){
console.info(node)；
})；
})；
```

下一个要注意的方法是filter()。filter对数组进行处理，该方法返回true的数组子集。例2-20中的迭代函数接收与map函数相同的参数，但应返回一个布尔值。

例2-20：Filter函数

```
[1，2，3，4，5].filter(even)；=>[2，4]
```

如果你想知道一些因素是否对数组的所有元素都为真，用ev ery()方法。它将对数组应用一个方法，如果该方法对数组中的所有元素都返回true，则返回true。当第一次出现false后停止。

如果你想知道有些条件是否对列表中的至少一个元素为真，可以使用some()方法。如果列表中至少有一个元素返回true，则将返回true。与every()方法一样，它将只评价得到结果的元素数量（当第一次出现true后停止）。

JavaScript的数组代数中的最后两个操作是reduce()和reduceRight()。第一种方法把一个数组简化成一些简单的值。这对一个累加器是非常有用的。在这种情况下，调用函数也接收累加值。因此，代码中使用reduce()对列表求和，就会如例2-21所示。你可以提供一个可选的初始值，作为第一次迭代之前的值进行传递。如果不这样做，开始时将使用数组的前两个元素。

例2-21：Reduce函数

```
[0，1，2，3，4，5].reduce(function(prev,current){
return prev+current;
},
initialValue);
```

如果JavaScript数组代数不提供完成任务所需要的方法，使用Array.prototype对象来创建它。如果有一个数字列表需要创建该列表的一个标准偏差，例如，可以简单地套用一个标准差的方法。在例2-22中为数组原型增加了一个标准偏差方法。

例2-22：标准偏差

```
var stdDev=[1，2，7，2....].stddev();
Array.reduce.sum=function sum(){
var sum=this.reduce(function(previous,current){
return previous+current;
});
return sum;
};
Array.prototype.square=function squareArray(){
return this.map(function(x){
return x*x;
});
};
Array.prototype.mean=function mean(){
return this.sum()/this.length;
};
Array.prototype.standardDeviation=function standardDeviation(){
var mean=this.mean();
var int1=this.map(function(n){
return n-mean;
});
var int2=int1.square();
var int3=Math.sqrt(int2.sum()/mean.length);
};
```

```
//给它起个短点的名字
Array.prototype.stddev=Array.prototype.standardDeviation;
```

你也可以扩展对象

如果你喜欢map函数而且希望把它用在你的对象上，也可以在代码中添加它。可以创建一个map函数，不仅能访问JavaScript对象中的所有节点，而且能将函数递归应用到它的每个子节点上。对于把数据结构转换成某种形式的节点树来说，这可能非常有用。例2-23检查用户的浏览器是否已经为指定对象定义了map()和filter()，然后如果有必要的话定义该函数。

例2-23：将map和filter扩展为对象

```
if(Object.prototype.map===undefined){
Object.prototype.map=function(fn){
var newObj={};
for(var i in this){
if(this.hasOwnProperty(i)){
newObj[i]=fn(i,this[i], this);
}
}
return newObj;
};
}
if(Object.prototype.filter===undefined){
Object.prototype.filter=function(fn){
var newObj={};
for(var i in this){
if(this.hasOwnProperty(i)){
if(fn(i,this[i], this)){
newObj[i]=this[i];
}
}
}
return newObj;
};
```

```
}
```

JavaScript对象可以成为现代应用程序中任意复杂的树，能够使用在文件系统中指定路径类似的方法找到特定对象的一个子树。其实，这非常容易实现。

path方法的路径形式与UNIX文件路径相同，如"/path/to/our/data"。它使用一个内部函数，以递归方式向下遍历数据树，直到找到请求的元素，并返回它。当发现元素不存在时，返回undefined。乍一看，只是调用了每个迭代的顶层路径函数，这种做法似乎有道理。但这不是一个好主意，因为有可能在某些情况下，路径中的一些点上可能会有一个有索引的数组，如果Array.prototype与Object.prototype不相同，将导致迭代被破坏。通过在内层函数中进行搜索，可以避免这个问题。

例2-24所示的方法可以处理数组和对象。如果路径的一部分是一个数字、数组和对象，那么都可以用方括号标记寻址，例如“[4]”，则将取组中的第5个元素。

例2-24：通过Path选择

```
Object.prototype.path=function FindByPath(path){
var elementPath=path.split('/');
var findItter=function findItter(element,path){
//如果元素为空则直接忽略并继续查询
if(path[0]===''){
return findItter(element,path.slice(1));
```

```
}
if(element[path[0]]===undefined){
return undefined;
}
if(path.length===1){
return element[path[0]];
}
return findItter(element[path[0]], path.slice(1));
};
return findItter(this,elementPath);
};
```

第3章　测试JavaScript应用

测试驱动开发的软件开发方式在过去几年风靡一时。通过使用自动化且可重复的测试方法，可以帮助开发人员编写更加高质量的代码，并且确保新添加的代码不会影响原有的功能。一些支持者甚至认为测试样例要在实际代码之前编写。

在大多数服务器端开发平台中，测试已成为开发的一个关键因素。在PHP、Java、Ruby等开发环境中已经出现Solid测试框架。然而，在大多数这些语言中测试的标准方法对JavaScript不起作用。为什么？让我们来看看几个原因。

服务器端测试套件一般都只是在一组环境下的测试程序。如果一个REST服务是用Python开发的，那么测试人员可以用几个安全假设创建测试。例如，他们可能会知道，它会在Linux上运行的Python3.0，以及所有支持软件的特定版本。

Web应用程序的开发人员没有这种信心。用户将使用Firefox、Internet Explorer、Chrome、Safari以及Opera浏览器，而且每种浏览器都有好几个版本。测试套件必须能够跨浏览器和操作系统处理测试，因为它们中的每一个都略有不同。

差别主要由两个原因产生。首先，语言本身在不同的浏览器间就有差异。例如，Firefox支持关键字const，而Internet Explorer不支持。其次，许多HTML接口只存在于某些浏览器或浏览器版本中。本书中的许多JavaScript接口也只存在于某些浏览器中，测试必须能够适应这些差异并在需要的地方进行退化处理。

Java或者C方面的测试专家主要讨论一些单元测试、集成测试等测试方法。这些测试方法的原理是一样的，只是目的不一样。

单元测试是一种运行很快的小规模测试，一次测试一个功能。它们需要遵循的准则是，对于任何特定的函数、方法或接口，如果给定的输入是x那么程序应该执行的功能就是y。它们测试的是系统的基本逻辑。每个单元测试理论上应该只测试一个方法或者一小块代码。

集成测试是一种相对复杂的测试方法，它用于确认所有代码协同工作的时候能运行良好。它们倾向于遵循的原则是“如果用户点击了这个按钮，那么系统就做这个工作”。

然而，这些测试方法在JavaScript中似乎不像在其他语言中那么适用。在JavaScript中，QUnit（参见本章的"Qunit"一节）更加适用于单元测试，而Selenium（参见本章的"Selenium"一节）则更适用于集成测试。然而，由于JavaScript中趋向于使用很多小的匿名函数，所以造成很难运行单元测试，因为这些函数很难被测试代码调用。一种辅助测

试的方式是在这些函数的外面创建很多功能一样的函数，可以把它们放在更大的命名空间里面，也可以把它们作为可以测试到的函数的返回值。

在例3-1中，makeInList用于测试传入的参数list中是否包含记录中的某个字段field。这种情况下，返回的函数是一个无副作用的函数，很容易被测试。例3-2展示了测试使用makeInList返回的函数两个方法。如果传入的是"NY"，函数将返回true，因为创建的列表中含有"NY"。反之传入"CT"就会返回false。

例3-1：In-list测试

```
var makeInList=fuction(list,field)
{
return function inList(rec)
{
var value=rec[field];
return list.indexOf(value)!==-1;
};
};
```

例3-2：使用In-list测试

```
var nynj=makeInList(['NY', 'NJ'], 'state');
ok(nynj({state: "NY"}));
ok(!nynj({state: "CT"}));
```

JavaScript的运行时模型也越来越复杂。在服务器端语言中，一个单元测试一般由几个连续的步骤组成：

1.设置所需的测试套件。

2.运行要测试的方法。

3.根据一些标准测试该方法的结果。

测试类型

测试人员经常谈论的几种测试包括单元测试、集成测试等。这些测试的基本工具是一样的，只是测试的层次不一样。

单元测试的目的是测试一小块的代码，经常是函数或者方法。比如，例3-2就是一个单元测试。它仅仅测试了一个函数，并且检查了这个函数所有可能运行到的情况。

集成测试用来确保整个系统按照之前设计的方案运行。集成测试可以是在浏览器中点击一个按钮，然后验证数据库的记录是否被更新，从而验证整个系统是不是可以协同工作。集成测试一般运行得比单元测试慢，所以，许多人实时运行单元测试，在晚上才进行集成测试。

验收测试用户确认软件是不是满足客户的需求。这个测试的主要目的不在于技术上，而是确保软件按照程序员的设计工作，因为这样才能满足用户的需求，从而进行部署。

然而，这种模型不适合JavaScript。在这种模型中，行为可以不是实时的，但是一段时间后可能发生。在JavaScript中，这样的测试会失败，因为在方法运行和它的结果准备好之间可能有延迟。在JavaScript中，测试可能更像下面这样：

1.设置所需的测试套件。

2.运行要测试的方法。

3.等待一个Ajax调用完成。

4.为动作的结果检查DOM（包括页面以外部分的不良影响）。

此外，还有一个障碍，在许多情况下要运行的方法是DOM元素上的一个回调。为了在这种情况下运行单元测试，需要找到该DOM元素，并给它正确的事件。这种调用处理程序的方式与实际用户使用该应用时的方式尽可能相似。甚至还不够：它不能通过DOM中的各种复杂的接口再现用户的点击体验。

基于浏览器的应用程序在测试用户界面时也比较复杂。因为大多数的测试驱动开发都是在服务器端完成，或者是测试数据处理的代码，这些测试的输入输出相对比较固定。对于给定的函数，输入"A"和输入"B"返回值都是确定的。

但是JavaScript程序由于更加关注用户界面而变得更加复杂，因为可能的用户输入行为序列变得非常庞大。所以需要测试的数目就非常大，甚至开发者都不能想到所有的情况。比如，如果用户在输入框输入名字的时候输入了重音符号怎么办？这时候系统还能正常反馈吗?

QUnit

QUnit是由制作jQuery的同一个团队开发的JavaScript测试套件。在QUnit中创建一个测试套件，用测试套件的名称和将要实际运行测试的函数两个参数调用测试函数。

注意：也可以配置QUnit来检查一个正在运行的测试是否引入任何新的全局变量，由于全局变量是JavaScript错误的主要来源之一，因此这是一个非常有用的功能。为了测试全局变量漏洞，开始测试时向URL添加"?noglobals"。

JSLint的使用（见附录：JSLint）也有助于发现全局漏洞和很多其他错误。可以在一组JavaScript文件上运行JSLint作为自动测试套件的一部分。

简单示例

以一个非常普通的Ajax应用为例。在例3-3中，有一个带有一个按钮的HTML页面。当单击按钮时，JavaScript的handleButtonClick()函数将一个Ajax查询发送给服务器，请求获得一个文件（在这种情况下，它是一个静态HTML文件），然后将该文件放在页面上的＜div＞中。请注意，为简单起见，把JavaScript直接放在文件中。一个清理器将JavaScript保持在一个单独的随测试一起加载的文件中。

为了测试这一点，需要运行该按钮的回调并验证结果是正确的。有两种方法可以做到这一点。测试程序可以发送一个单击事件给按钮，或者直接调用函数。对于QUnit中的测试，直接调用函数更简单，但要求把函数绑定到一些变量上，这些变量可以通过测试工具看到。本例中已经这样做了，将函数赋值给变量handleButtonClick。调用回调后，测试函数进行短暂等待，让Ajax调用运行，然后用jQuery检查元素是否存在。

例3-3：简单应用

```
   ＜!DOCTYPE html＞
   ＜html＞
   ＜head＞
   ＜script src="http://code.jquery.com/jquery-latest.js"＞＜/script
＞
   ＜script src='button.js'＞
   ＜/script＞
   ＜script＞
   var handleButtonClick=function handleButtonClick(){
   $().get('document.html', '', function(data,status){
   $("＜div
＞").attr({id: 'target_div'}).text(data).appendTo('body');
```

```
    });
    };
    $('button.click_me').click(handleButtonClick);
    </script>
    <link rel="stylesheet"
    href="http://github.com/jquery/qunit/raw/master/qunit/qunit.css"
    type="text/css"
    media="screen"/>
    <script
src="http://github.com/jquery/qunit/raw/master/qunit/qunit.js">
</script>
    <script src='simple-test.js'></script>
    </head>
    <body>
    <button id='click_me'>Click Me</button>
    <!--QUnit要求的内容-->
    <h1 id="qunit-header">QUnit example</h1>
    <h2 id="qunit-banner"></h2>
    <h2 id="qunit-userAgent"></h2>
    <ol id="qunit-tests"></ol>
    <div id="qunit-fixture">test markup,will be hidden</div>
    </body>
    </html>
```

QUnit测试工具被页面加载到加载器上，像其他要加载的JavaScript程序一样。本例从http://github.com/jquery/qunit/raw/master/qunit/qunit.js加载。该文件将开始运行加载器上的任何jQuery测试。一旦测试运行完毕后，这些测试结果将显示在页面上，这是QUnit为何需要DOM中存在这些元素的原因。

为了测试这个例子，在例3-3中测试工具将加载QUnit，然后调用handleButtonClick方法。例3-4通过传递值1000给setTimeout方法，等待一秒钟直到文件被加载。一秒之后，测试＜div＞在DOM中是否存在

（第一次调用equal），从div得到文本，并检查文本中的第一个单词是否为"First"，这是预期值（第二次调用equal）。可以选择一个更完整的测试，每四分之一秒检查一次该元素，直到它出现或到达最大时间限制。在现实世界中网页加载时间可能有所不同，这取决于外部因素，包括网络的使用和服务器负载。QUnit测试是一个由测试工具调用的JavaScript函数。请看例3-4中的简单测试。测试使用多个要通过测试必须满足的断言函数。为了测试一个值是否等于预期的结果，可使用equal()方法，它有三种方式，测试值、预期的结果和参数选项，这是一个消息，用于显示测试是否失败。使用此消息将有助于弄清楚测试失败的对象。如果在代码写完6个月后才测试代码，则更为有用。

例3-4：简单测试

```
test("Basic Test", function(){
//判断目标属性不存在
equal($('div#target_div').length, 0,
"Target element should not exist");
//运行方法
handleButtonClick();
equal($('div#target_div').length, 0,
"Target element should still not exist");
window.setTimeout(function(){
start();
equal($('div#target_div').length, 1,
"Target element should now exist");
equal("First",
$('div#target_div').text().substr(0, 5),
"Check that the first word is correct");
}, 1000);
stop();
});
```

用QUnit测试

要运行的QUnit测试必须包含QUnit样式表和JavaScript文件，这些可以直接从Github拖入或从本地加载（见例3-3）。DOM还必须包括几个元素，让QUnit使用以显示其结果。在例3-3中的HTML下方可以看到。这是运行测试需要的所有条件。

QUnit提供了八个断言函数。除了equal()函数（它出现在我们前面的例子中）外，还有对等式和ok()方法的进一步测试，该方法测试传递给它的值是否为真。还有根据JavaScript“===”操作符的strictEqual()测试，而equal()使用“==”操作进行比较。

为了测试一个更复杂的数据结构是否是相同的，使用deepEqual()。它对两个数据结构作递归比较。

每个等式函数都有一个相反的形式，可以测试等式的缺陷：notEqual()、notStrictEqual()和notDeapEqual()。equal的各版本都用相同的参数，但测试相反的情况。

最后的判断是raises()，以一个函数为参数，并期望抛出错误情况。

要测试异步方式发生事件，使用返回值无法正常工作。在这种情况下，测试必须等待动作完成。可以通过用setTimeout()设置一个超时来实现，当设定的时间到达后再运行。或者可以用回调如Ajax加载或其他事件来实现。

Selenium

虽然QUnit允许测试JavaScript代码，但是Selenium（http://seleniumhq.org/）采用了不同的方法。Selenium通过模拟用户可能会采取的行为测试用户界面。一个Selenium测试包括一些浏览器中的运行步骤，例如加载一个页面、点击一个特定的元素、输入文字到一个文本区等。这些行为夹杂着判断，验证要测试的DOM状态或其他东西。其中可能包括对元素或文本是否存在的测试。

当Selenium测试运行时，实际上将启动一个浏览器，并用或多或少与用户操作相同的方式运行它。因此可以通过浏览器监视该测试的交互。甚至可以在测试运行的同时手动与浏览器交互（虽然这可能不是一个好主意）。

Selenium由几个大多独立的部分组成。其一是作为Firefox浏览器插件实现的IDE。另一个是Selenium RC服务器seleniumrc，它是一个可用于在不同的浏览器上自动运行测试的Java服务器。

Firefox的Selenium IDE插件是开发人员最好的朋友。它允许构建直接在浏览器中运行的测试。IDE可以记录用户的操作，并稍后作为一个测试回放。它还可以让你按步调试一个测试（每次一行），这对在测试中发现时序问题非常有用。

注意：在默认情况下，录制动作时Selenium IDE将使用各种HTML元素的ID。在程序员没有明确分配ID的情况下，一些框架将会在元素创建时顺序地分配ID。这些ID每次运行都不相同，所以请使用一些其他的方法标识感兴趣的元素。

Selenium IDE将测试输出为HTML文件，可以在Firefox的IDE自身中运行。此外，这些HTML文件可以作为批处理任务在Selenium RC组件中运行。Selenium RC服务器也允许对运行于任意浏览器上的HTML进行测试，因此可以在IE、Chrome、Opera或Safari上运行这些测试。Selenium RC服务器也可以由传统测试通过像phpUnit或jUnit一样运行的测试进行控制。

Selenium IDE在开发中记录Web宏方面也很有用。例如，如果正在测试Web应用中的一个向导，在要调试的界面之前会显示四个或五个界面，可以使用IDE来创建Selenium脚本，然后把它作为一个宏调用，从而可以自动地找到正在测试的点。

注意：如果一个测试在单步模式下运行时起作用，但工作不正常，它可能需要几个暂停语句让浏览器赶上来，或者一些waitFor……语句更好，这将使测试和浏览器同步。

Selenium中有三种方式来运行测试，通过Selenium IDE、用Selenium RC的测试工具以及用一种编程语言。IDE在交互环境中容易

使用，但只在Firefox中有用。测试工具接受HTML格式的输入测试，它可以在IDE中创建。最后，可以在一个单元测试框架中如phpUnit，用编程语言编写测试。使用基于测试工具或编程语言的测试套件允许对浏览器进行全套测试，并能提供报告和其他功能。这个过程也可以整合到持续集成工具，以及用xUnit框架编写的任何其他测试中。

一个Selenium测试由一个包含一个表的HTML文件构建而成，测试中的每一步是表中的一行。该行包括三列：要运行的命令、要操作的元素和在某些情况下使用的参数选项。例如，第三列包含测试表单时要输入到一个input元素中的文本。

与QUnit测试不同，Selenium测试都是关于用户界面的测试。因此，Selenium是比单元测试更集成的测试。为了用Selenium测试例3-3，需要与QUnit不同的方法。QUnit测试直接调用处理函数，而Selenium测试点击按钮，并等待＜div＞显示。

例3-5完成该测试。表中的每一行作为测试的一部分执行一个操作。第一行打开的网页进行测试，然后第二行单击按钮（通过元素的ID识别）。然后，测试等待页面显示＜div＞，在这种情况下，通过XPath进行识别。

此测试对上面QUnit测试的同一个简单脚本进行测试。但不是测试函数本身，而是用与人工测试员类似的方法测试用户界面。它打开页

面，单击click_me按钮。然后等待div元素target_div在DOM中出现。然后，它宣布单词"First"出现在该页面中。

例3-5：简单Selenium测试

```
    <?xml version="1.0"encoding="UTF-8"?>
    <!DOCTYPE html PUBLIC"-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
    <html xml：lang="en"lang="en">
    <head profile="http://selenium-ide.openqa.org/profiles/test-
case">
    <meta http-equiv="Content-Type"content="text/html；charset=UTF-
8"/>
    <link rel="selenium.base"href="http://www.example.com/"/>
    <title>New Test</title>
    </head>
    <body>
    <table cellpadding="1"cellspacing="1"border="1">
    <thead>
    <tr><td rowspan="1"colspan="3">New Test</td></tr>
    </thead><tbody>
    <tr>
    <td>open</td>
    <td>/examples/simple.html</td>
    <td></td>
    </tr>
    <tr>
    <td>click</td>
    <td>click_me</td>
    <td></td>
    </tr>
    <tr>
    <td>waitForElementPresent</td>
    <td>//div[@id='target_div']</td>
    <td></td>
    </tr>
    <tr>
    <td>assertTextPresent</td>
    <td>First</td>
    <td></td>
    </tr>
    <!--More tests-->
    </tbody></table>
```

```
</body>
</html>
```

Selenium命令

Selenium拥有丰富的命令语言——Selenese（http://seleniumhq.org/docs/04_selenese_commands.htm），允许程序员创建测试。用户在浏览器中进行的几乎任何动作都可以用Selenium命令实现。可以通过创建一个包含一系列动作和测试的脚本来创建Selenium测试。

警告：从桌面拖动一个文件到浏览器（见第6章的“拖曳”），不能用Selenium实现，也不能简单地用qUnit测试。

有极少数例外，Selenium命令将DOM中的要操作的元素的位置作为一个参数。该位置可以在几种方法中的一个进行指定，包括元素ID、元素的名称、XPATH、CSS类、DOM内部的JavaScript调用和链接文本。第3章的“Selenium位置选项”对这些选项进行了说明。

警告：在ExtJS中使用元素ID没有效果，因为ExtJS每次分配给元素的ID会有变化。请使用CSS类或者HTML元素的其他属性。为表示按钮，使用XPath中按钮的文本，像//button[text()='Save']通常很有用，也可以选择使用属性如：//img[@src='img.png']。

注意：Selenium基本命令不包括任何条件或循环的能力。一个Selenium HTML文件顺序地从上到下运行，当断言失败或最后一个命令执行时结束。如果需要流程控制，请使用goto_sel_ide.js插件（http://51elliot.blogspot.com.2008/02/selenium-ide-goto.html）。

这个插件对查找内存泄漏或其他问题很有用，这些问题可能会出现在用户长时间运行的应用程序中。要摆脱内存泄漏，JavaScript仍然有很长的路要走，当页面重载频繁，JavaScript和DOM状态重置时内存泄露不是一个问题。

Selenium中大量的命令可以用来构建测试。Selenium IDE包含一个命令的参考，一旦学会一些基本知识，就可以很容易地对于任何给定的情况找出正确的命令。表3-1显示了一些常见的Selenium命令。它们往往在两个基本组中：动作和断言。动作包括click、type、dblclick、keydown、keyup等。断言提供实际测试，能让Selenium找出用户的行为如何影响页面。断言可以暂停脚本，但不会产生页面变化。

表 3-1：选定的Selenium命令

命令	目标	动作
open	要打开的Web页面	打开一个Web页面
dblclick	要双击的对象	双击一个元素
click	要单击的元素	单击一个元素
mouseOver	鼠标移动处的元素	复制一个mouseOver事件
mouseUp	鼠标按键弹起处的元素	复制一个mouseUp事件
mouseDown	鼠标按键按下处的元素	复制一个mouseDown事件
type	用XPath或其他选择器选择的元素，第三列是要输入的文本	模拟文字输入

表 3-1：选定的Selenium命令（续）

命令	目标	动作
windowMaximized		最大化当前窗口
refresh		刷新浏览器。可以于重置JavaScript状态
dragAndDrop	到拖动元素的偏移，选择器是要拖动的元素。	拖曳一个元素

Selenium位置选项

Selenium提供了6种定位网页元素的方式。选择正确的定位方式会简化开发测试代码的复杂度：

ID

提供一个HTML ID：

```
id
```

Name

提供一个元素名（可用于表单输入）：

```
name=username
```

```
XPath
```

用XPath找到一个元素：

```
//form[@id='loginForm']/input[1]
```

CSS

通过CSS Selector找到一个元素，用户比较熟悉的过程是jQuery,Selenium中的CSS Selector引擎比jQuery的更有限：

```
css=div.x-btn-text
```

Document

用DOM找到一个元素：

```
dom=document.getElementById('loginForm')
```

Link text

通过href属性中的文本找到元素（可用于HTML链接）

```
link='Continue'
```

注意：在ExtJS或任何其他自定义的部件中，如果一个click事件按预期工作，尝试使用mouseDown。例如，要在表格中选择一行，用mouseDown事件，而不是click事件。当用鼠标单击时，浏览器发送三个事件：mouseDown、mouseUp和click事件。用户界面中的不同元素可能响应其中任何一个。

Selenium中的动作都有两种形式：一种简单形式和一种将等待页面重新加载的形式。点击命令的等待形式，例如clickAndWait。

经过一系列动作，有必要验证该应用实际上执行了正确的动作。Selenium中的测试大多数对元素存在与否进行测试。例如，测试向一个ExtJS表格中添加一个新元素，测试脚本会做这样的事情：

1.点击添加按钮。

2.填写表格，提供默认值。

3.向服务器提交新的记录。

4.等待服务器的响应，并验证该文本在正确的表格中。

所有的断言有三种基本形式：基本形式、校验形式和WaitFor形式。基本的命令类似于assertElementPresent，并且当断言失败时将停止测试。verifyElementPresent将检查元素是否存在，如果不存在则继续测试。如果有多个测试，又不希望它们在一次失败后停止，这将非常有用。如果一个动作应该有一个延迟的结果，则用waitForElementPresent，这将暂停测试脚本，直到条件满足或到达测试时间。小结：

assert...

检查某事是否为真，如果不为真则停止。

verify...

检查某事是否为真，如果不为真则继续。

waitFor..

等待页面上的某事发生（往往用Ajax）。

用Selenium IDE构建测试

Selenium测试可以手工构建，但自动往往更容易。Selenium IDE的Firefox插件可以在浏览器中记录动作，并把它们保存为测试。程序员仍然必须加入断言、手动等待命令，并可能要调整已产生的脚本。测试被保存为一个HTML文档，可以在版本控制内部进行检查并从IDE和自动测试工具中运行。

IDE是一个在Selenium中测试选项的不错方式。它也可以创建测试脚本并直接从IDE中运行它们。Selenium IDE还允许控制执行脚本的快慢程度，并通过它们进行单步调试。左侧面板（在图3-1被隐藏了）显示了一个所有已定义测试用例的列表。它们都可以通过工具栏上的第一个按钮运行（速度控制的左侧）。右边的下一个按钮执行一个测试。

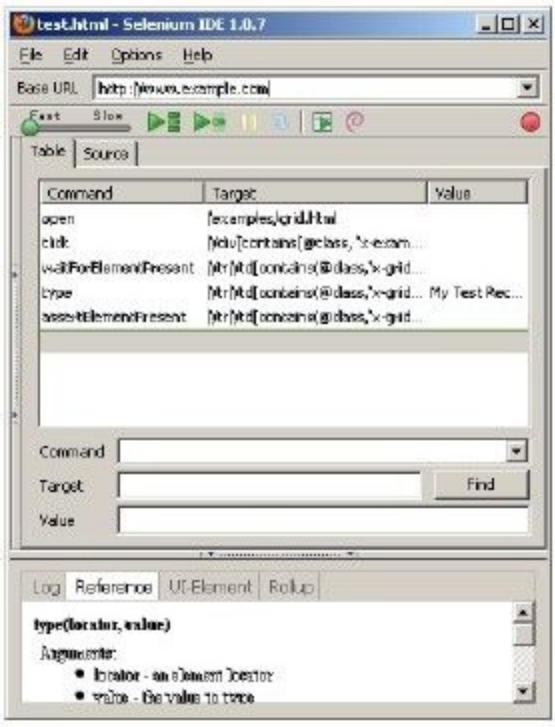

图　3-1　Selenium IDE

Selenium IDE底部的面板是功能选项卡（可以用插件添加更多）。最左边的选项卡功能是正在运行测试的日志。第二个选项卡是参考面板。当从中间面板的“菜单”选择一个命令时，此选项卡将显示选定命令的信息，包括需要什么样的参数。

自动运行测试

可以用一个流行的测试套件（JUnit或PHPUnit）来运行Selenium测试，让测试跨多个浏览器和平台运行。如果正在定期运行一个单元测试，Selenium测试可以从正常测试中（见例3-6）运行。

注意：大部分或全部修改这里描述的PHPUnit后，它在所有其他语言类似的测试套件中可用。

这些测试将像任何其他的测试一样在测试环境运行。每个HTML文件将作为测试在测试套件中运行（细节可能取决于所使用的测试工具）。测试工具将顺序地以与IDE中类似的方式运行每个HTML文件。

要在PHPUnit中运行测试，需要设计一个或多个测试机来运行浏览器。每个测试机需要运行一个Selenium RC的程序副本，并安装有浏览器。seleniumrc的二进制文件是一个".jar"的java文件，因此能够在Windows、Linux或Mac操作系统上运行。

当在PHPUnit中运行测试时，测试类PHPUnit_Extensions_SeleniumTestCase将联系seleniumrc程序，并要求它启动一个浏览器实例，然后在一个REST接口上发送命令给它。

如果在"$browsers"的静态成员中或通过"phpunit.xml"文件（见例3-7）列出了多个浏览器，"PHPUnit_Extensions_SeleniumTestCase"类将顺序地运行，为每个浏览器测试运行每个测试。例如，在例3-6中，将在Safari、Firefox、Chrome和Internet Explorer上运行测试。在phpunit.xml文件中列出浏览器选项更好，因为可以创建多个文件，以帮助实现不改变测试源代码即可修改测试选项。

下面的测试从文件seleneseTest.html运行一个Selenium测试。但是，通过设置测试类的$seleneseDirectory属性为该文件的路径，可以让它自动运行Selenium HTML测试文件的整个路径。

```
public static$seleneseDirectory='/path/to/files';
```

例3-6：phpUnit中运行Selenium

```
<?php
require_once'PHPUnit/Extensions/SeleniumTestCase.php';
class WebTest extends PHPUnit_Extensions_SeleniumTestCase
{
protected$captureScreenshotOnFailure=TRUE;
protected$screenshotPath='/var/www/localhost/htdocs/screenshots'
;
protected$screenshotUrl='http://localhost/screenshots';
public static$browsers=array(
array(
'name'=>'Safari on MacOS X',
'browser'=>'*safari',
'host'=>'mac.testbox',
'port'=>4444,
'timeout'=>30000
),
array(
'name'=>'Firefox on Windows',
```

```
'browser'=>'*firefox',
'host'=>'windows.testbox',
'port'=>4444,
'timeout'=>30000
),
array(
'name'=>'Chrome on Windows XP',
'browser'=>'*googlechrome',
'host'=>'windows.testbox',
'port'=>4444,
'timeout'=>30000
),
array(
'name'=>'Internet Explorer on Windows XP',
'browser'=>'*iexplore',
'host'=>'windows.testbox',
'port'=>4444,
'timeout'=>30000
)
);
protected function setUp()
{
$this->setBrowserUrl('http://www.example.com/');
}
public function testSeleniumFile()
{
$this->open('http://www.example.com/');
$this->runSelenese('seleneseTest.html');
}
}
?>
```

例3-7：phpunit.xml

```
<phpunit stopOnFailure="true"
verbose="true"
strict="true">
<php>
</php>
<selenium>
<browser name="Firefox"
browser="*firefox"
host="192.168.0.10"
port="4444"
timeout="30000"/>
```

```
＜browser name="Chrome"
browser="*chrome"
host="192.168.0.10"
port="4444"
timeout="30000"/＞
＜!--其他浏览器--＞
＜/selenium＞
＜testsuites＞
＜testsuite name="Selenium"＞
＜file＞/path/to/MyTest.php＜/file＞
＜/testsuite＞
＜/testsuites＞
＜/phpunit＞
```

相对于Selenium RC的所有优势，它有一个主要的缺点：每次只能运行一个测试。因此，如果用一个大的测试套件和大量不同的浏览器运行该测试，完整的测试运行可能花费好几个小时。Selenium Grid为此提供了解决方案，允许在一组机器上并行运行大量测试。可以在http://selenium-grid.seleniumhq.org/找到Selenium Grid软件的例子和文档。

如果不想建立测试场，网上有"cloud selenium farms"可用。此外，可以在亚马逊的EC2云服务中运行Selenium。对于偶尔使用的用户或没有资源用于建立和维护一个本地selenium测试场的初学者，这是非常有用的。对于查看应用将如何在远程网络中执行也非常有帮助。

Selenese命令编程接口

Selenium测试套件可以从一个HTML文件或直接从单元测试代码中运行一个测试。Selenium RC服务器也有一个API可以在几种语言中通过单元测试代码进行调用，可以写一个PHP、Ruby、Python、PERL、JAVA及C#/.NET的Selenium测试。可以通过Selenium IDE或手动创建代码的测试用例。Selenium IDE将生成一个测试框架。要做到这一点，在IDE中记录测试，然后为运行测试的语言选择输出选项，则测试将转换成该语言。

注意：要在多个浏览器上运行Selenium，需要设置Selenium服务器。见第3章的“Selenium RC及一个测试场”。

直接从单元测试运行Selenium，宿主语言作为语言拥有全部力量，尤其是它的流程控制，而HTML样式测试则有限得多。通过在服务器端编程语言使用的API，使得为编写Web脚本创建一个非常丰富的环境成为可能，当然，必须访问服务器端的库检查数据库中的数据或者访问Web服务，可以构建一个测试，在浏览器中执行一些动作，然后在数据库或日志文件中对结果进行检查。

使用服务器端测试的另一个好处是，如果正在使用任何形式的持续整合如CruiseControl或phpUnderControl,Selenium测试就像一个测试

系统，只是可以做更多测试，而且不论团队正在使用的是什么语言都能测试。在一个使用了测试框架的团队中，这将充分利用团队的现有经验。

例3-8是用PHP在PHPUnit测试框架下写的一个很简单的Selenium测试的例子。它首先打开一个页面，等待页面载入完毕后，判断页面标题是不是"Hello World"，然后它会点击"Click Me"按钮。如果标题不是"Hello World"或者没有"Click Me"按钮，测试就会返回失败。

例3-8：测试Hello World

```
<?php
require_once 'PHPUnit/Extensions/SeleniumTestCase.php'
class WebTest extends PHPUnit_Extensions_SeleniumTestCase
{
functiontestTitle()
{
$this->open('http://www.example.com');
$this->assertTitle('Hello World');
$this->click("//button[text()='Click Me']");
}
}
```

Selenese命令一般包括动作、断言和查询。动作通常反映Selenium动作的基本形式：click、mouseOver、mouseDown、type等。各种出现在Selenium IDE中的延迟形式是不存在的，但可以很容易地通过使用sleep函数的循环进行模拟。

许多在HTML Selenese测试工具中出现的实用方法在服务器端Selenese接口中并不存在，不包括WaitForElementPresent（见例3-9）或WaitForElementNotPresent方法在内，但可以很容易地通过为测试创建自定义基类进行添加。

例3-9：WaitForElementPresent

```
function waitForElementPresent($el，$timeout=60)
{
While($timeout)
{
if($this-＞isElementPresent($el))
{
return true；
}
$timeout-=1；
sleep(1)；
}
$this-＞fail("Element$el not found")；
}
```

查询允许程序员确定行动发生后页面的状态。这些查询允许程序员找出页面上是否存在一个元素或一段文字，或了解页面的当前状态。

Selenese API还提供了许多方法，从一个单元测试环境中的HTML文档中获取数据。为了测试页面中是否存在一个给定文本，请使用"$this-＞isTextPresent()"方法，这对检查一个文本是否存在往往非常有用。要找出一个元素是否存在，请使用$this-＞isElementPresent()方法。要真正从DOM中获取文本，请使用$this-＞getText()方法。它对

Selenese支持的任何形式的选择器进行处理，并返回该元素的文本。如果该元素不存在，该方法将抛出一个异常。

XPath选择器匹配页面上的多个元素。它通常会返回匹配的第一个元素。要知道有多少个元素匹配，请使用$this->getX pathCount()方法。

当在PHP中建立测试时，对浏览器中的接口与运行在PHP中的测试本身之间的测试动作进行同步可能是一个挑战。这很直观，写一个进行了某些操作的测试，例如单击一个元素，然后立即移动鼠标到另一个元素上。这不可避免地会失败，因为JavaScript将需要花费一段时间（也许是0.1s），以创建一个用户界面或等待数据从服务器加载。有两种方式来处理这个问题。

简单的方法是在PHP代码中加入延时。一些放在正确位置的sleep()命令，可以正常地测试函数。然而，可能会导致必须花费较长的时间运行测试。为了加快测试，使用如waitForElementPresent()这样的方法在测试之间延时1/10s，可以让脚本运行速度更快，只要DOM中有一种方法能判断浏览器为下一步测试做好准备。

第二种方式是在Selenese接口中使用$this->getEval()方法评估自定义JavaScript。向该方法传递一个字符串，其中包含要执行的JavaScript。当通过getEval()调用JavaScript时，它将运行在测试工具窗

口的窗口背景下，而不是测试窗口中。因此，全局变量必须以全局窗口对象为前缀（通常不要求）。

在例3-10中，Selenium在JavaScript中执行getEval()获取全局变量session_id。

例3-10：用Selenese运行JavaScript

```
$session_id=$this->getEval('window.session_id');
```

注意：也可以使用Selenium RC设置手动测试，通过让脚本打开该页面，并执行所有步骤，直到到达需要测试的点。当到达终点时，它应该长时间的暂停，因为测试结束时浏览器将关闭。在测试停止的点，可以由一个人接管浏览器，并执行任何可能需要的手动操作。当开发一个多步向导或类似的用户界面时，这往往是有益的。

在Selenium中运行QUnit

Selenium能很好地运行QUnit测试。要实现这一点，只需要在Selenium中加载QUnit页面并运行测试。也可以选择只通过传递参数到URL字符串运行测试的一个子集。通过整合Selenium与QUnit，可以把QUnit中的浏览器测试结果导出到持续整合的测试工具中。

Selenium只打开QUnit的URL，然后退到后台，等待测试完成。为了让测试工具知道测试是否通过或失败，QUnit提供了一个简单的微格式（见例3-12），显示运行了多少测试，有多少通过或失败。然后，单元测试可以通过XPath选择器查看此数据，并确保所有测试通过。

在例3-11中，PHP程序打开本章开头显示的QUnit测试，然后等待测试运行。当测试完成后，qunit-testresult元素将插入到DOM中。在这一点上，Selenium可以找到运行测试的数量以及有多少测试通过或失败。例3-13展示了用PHP代码从Selenium中提取QUnit返回值的方法。

例3-11：运行QUnit的Selenium测试

```
<?php
require_once 'PHPUnit/Extensions/SeleniumTestCase.php';
class WebTest extends PHPUnit_Extensions_SeleniumTestCase
{
function testQUnit()
{
//这里在QUnit测试中添加你的HTML文件的URL
```

```
    $this->open('http://host.com/simple.html');
    $this->waitForElementPresent("//p[@id='qunit-testresult']");
    $failCount=$this->getText("//p[@id='qunit-
testresult']/span[@class='failed']");
    $passCount=$this->getText("//p[@id='qunit-
testresult']/span[@class='passed']");
    $totalCount=$this->getText("//p[@id='qunit-
testresult']/span[@class='total']");
    $this->assertEquals($passCount, $totalCount,
    "Check that all tests passed$passCount of$totalCount passed");
    $this->assertEquals("0", $failCount,
    "Checking result of QUnit tests$failCount/$totalCount tests
failed");
    }
    function waitForElementPresent($element, $timeout=60)
    {
    $time=0;
    while(!$this->isElementPresent($element))
    {
    $time++;
    if($time>$timeout)
    {
    throw New Exception("Timeout: $element not found!");
    }
    sleep(1);
    }
    }
    }
```

例3-12：QUnit结果的微格式

```
    <p id="qunit-testresult"class="result">
    Tests completed in 221 milliseconds.<br/>
    <span class="passed">1</span>tests of
    <span class="total">2</span>passed,
    <span class="failed">1</span>failed.
    </p>
```

例3-13：从QUnit获取数据的PHP代码

```
    <?php
```

```
   $failCount=$this->getText("//p[@id='qunit-
testresult']/span[@class='failed']");
   $passCount=$this->getText("//p[@id='qunit-
testresult']/span[@class='passed']");
   $totalCount=$this->getText("//p[@id='qunit-
testresult']/span[@class='total']");
   $this->assertEquals($passCount, $totalCount,
   "Check that all tests passed$passCount of$totalCount passed");
   $this->assertEquals("0", $failCount,
   "Checking result of QUnit tests$failCount/$totalCount tests
failed");
```

Selenium RC及一个测试场

重要的是要确保应用不只在一个浏览器上运行良好，而是在多数的浏览器和平台上运行良好。在大多数情况下，可以假设应用可能会运行在Windows XP、Windows Vista、Windows 7、Mac OS X以及Linux平台上。此外，用户可能使用Firefox、Chrome、Internet Explorer、Safari和Opera浏览器，而且各浏览器可能有多个不同的版本。

在这些不同的浏览器中，JavaScript和各种接口的实现是类似的，并使用了一个与jQuery类似的框架消除一些差异，但浏览器仍然不完全相同。因此，一段在Firefox中行之有效的代码可能会在Chrome或Safari中突然崩溃。当然，代码可能将在浏览器的一个版本中有效，但在较旧的或较新的版本中没有效果。因此，跨多种浏览器测试是至关重要的。然而，让一个QA团队人工完成它可能是期望过高的，因此需要自动化实现。

Selenium RC使得用机器网络测试所有这些不同的组成部分成为可能。每个测试机必须安装有Java。在许多情况下，Java将默认安装，但如果不是，下载并安装它。然后从http://seleniumhq.org/download/下载Selenium RC包并解压。在每个服务器上，用下面所示的命令启动

Selenium server jar。如果一台机器是测试场中的成员，让Selenium服务器随机器启动自动运行可能是一个好主意。通常情况下，Selenium服务器采用默认安装，它提供了一些允许某种程度自定义的命令行选项。具体来说，如果需要的话，可以用端口选项改变默认的4444端口。这可以用来在一台服务器上运行多个服务器实例，以便同时在多个浏览器上测试。

```
java-jar selenium-server.jar[-port 4444]
```

注意：没有必要为每个测试服务器使用单独的一台机器。任何常见的虚拟机技术都能处理得很好。一台有足够RAM的功能强大的服务器应该能够运行一个小型的实用虚拟测试场。

如果需要在Android或iOS上测试应用，Selenium也能做得很好。有一个Android的Selenium驱动程序（http://code.google.com/p/selenium/wiki/AndroidDriver），以及一个iPhone的驱动（http://code.google.com/p/selenium/wiki/IPhoneDriver）。请注意，为了运行iPhone的驱动程序，需要有一个Mac以及iPhone的开发设置。由于iPhone store中没有Selenium驱动程序，需要会用提供的配置文件在手机上安装它，或者使用Xcode中的模拟器。

第4章　本地存储

Web浏览器给JavaScript提供了一个良好的环境，允许它创建可以在浏览器上运行的应用。使用ExtJS或jQuery可以构建一个与桌面应用媲美的多用途应用程序，并提供与桌面应用一样简单的分布式方法。然而，浏览器在提供用户体验方面，当涉及数据存储时曾是一片空白。

从历史上看，浏览器没有通过任何方法存储数据。它们实际上是最终的瘦客户机。最接近的可能是HTTP cookie机制，允许在每个HTTP请求中附加一块数据。然而，cookie遇到许多问题。首先，每个cookie随着每个请求来回发送。因此，浏览器为每个JavaScript文件、图片、Ajax请求等发送cookie。这会无缘无故地增加大量的带宽使用。其次，试图创建的cookie规范，使不同的子域之间可以共享一个cookie。如果一家公司有app.test.com和images.test.com，可以设置一个cookie对二者都可见。出现这种问题的原因是因为在美国以外的国家，由三部分组成的域名成为普遍现象。例如，可以在co.il主机中设置一个cookie，允许cookie渗透到以色列的几乎所有主机上。不能每当域名中包含一个国家的后缀时，就简单地请求一个由三部分组成的域名，因为一些国家如加拿大不遵循相同的约定。

在浏览器上进行本地存储，可能成为速度方面的一大优势。正常的Ajax查询根据服务器的不同可能花费半秒到几秒的时间。然而，即使在最好的情况下，也可能相当缓慢。从我在特拉维夫的办公室到加利福尼亚的一个服务器，一个简单的ICMP ping花费的平均时间约为250ms。在这250ms中，有很大一部分可能是由于基本的物理限制：数据沿着导线传送，速度仅为光速的一部分。因此，只要数据必须在浏览器和服务器之间往返，就很难有办法提高访问的速度。

本地存储选项对静态的或大多是静态的数据是一个非常好的选择。例如，许多应用程序有一个国家的列表作为数据的一部分。即使该列表中包括一些额外的信息，例如产品是否在每个国家都提供，列表也不会频繁变化。在这种情况下，通常将数据预加载到本地存储对象中，然后在必要时有条件地重载，这样用户将获得新的数据，而不必等待当前数据。

当然，本地存储对于处理一个可以脱机工作的Web应用也是必不可少的。虽然近来似乎到处都可以对互联网进行访问，但这不是绝对的，即使对智能手机也是如此。使用移动设备（如iPod touch）的用户，只有在有WiFi的地方才能访问互联网，甚至像iPhone或Android智能手机也有不能上网的死区。

随着HTML 5的发展，已经发起了一系列运动，为浏览器提供一种创建一个持久的本地存储的方法，但这一运动的结果尚未形成。如

何在客户端存储数据，目前至少有三个不同的方案。

在2007年，作为Gears的一部分，Google推出基于浏览器的SQLite数据库。基于Webkit的浏览器，包括Chrome、Safari、iPhone和Android手机上的浏览器，实现了一个Gears SQLite数据库版本。然而，SQLite从HTML 5方案中去掉了，因为它是一个单源的组件。

localStorage机制提供一个持续跨Web重载的JavaScript对象。这种机制似乎是合理的、一致的和稳定的。localStorage对于存储小规模的数据，如session信息或用户偏好。

本章介绍如何使用当前的localStorage实现本地存储。在下面的章节中，将看到已经出现在一些浏览器中的更加复杂和先进的本地存储形式：IndexedDB。

localStorage和sessionStorage对象

现代浏览器为程序员提供两个存储对象，localStorage和sessionStorgage。每个对象都将数据保存为键和值。它们有相同的接口，以同样的方式工作，除了一个例外。localStorage对象持续跨浏览器重新启动，而sessionStorage对象在浏览器的session重新启动时对自身进行重置。这可能是当浏览器关闭或窗口关闭的时候。确切在什么时候发生这种情况将取决于具体的浏览器。

设置和获取这些对象是非常简单的，如例4-1所示。

例4-1：访问localStorage

```
//设置
localStorage.sessionID=sessionId;
localStorage.setItem('sessionID', sessionId);
//获取
var sessionId;
sessionId=localStorage.sessionID;
sessionId=localStorage.getItem('sessionId');
localStorage.sessionId=undefined;
localStorage.removeItem('sessionId');
```

浏览器存储如cookies实现了一个“同根同源”的策略，使不同的网站互不干扰或读取对方的数据。但在本节中的两种存储对象都存储在用户的磁盘上（与cookies相似），因此一个有经验的用户就能找到方法来编辑数据。Chrome的开发者工具允许程序员编辑存储对象，可以在Firefox中通过Firebug或一些其他工具编辑它。因此，尽管其他网站不能向存储对象中注入数据，但是这些对象仍然不应该被信任。

Cookies受到了一定的限制：Cookies的大小只限于约4KB，必须通过每个Ajax请求传送到服务器，大大提高了网络流量。浏览器的localStorage大方多了。HTML 5规范中没有对大小列出确切的限制，但大多数的浏览器为每个Web主机作了5MB左右的限制。程序员不应该假定一个非常大的存储区域。

数据可以通过直接访问对象或一组访问函数存储在存储对象中。Session对象只能存储字符串，因此存储的任何对象类型都要强制转换为一个字符串。这意味着一个对象将存储为[object Object]，这可能不是想要的结果。因此要存储一个对象或数组，先将其转换为JSON。

每当一个存储对象中的值发生变化时，它会触发一个存储事件。此事件将显示键、它原来的值以及它的新值。例4-2是一个典型的数据结构。不像某些事件（如点击），存储事件不能阻止。应用没有办法告诉浏览器不做出改变。变化发生后，事件只是简单地通知应用。

例4-2：例4-2存储事件接口

```
var storageEvent={
key：'key'，
oldValue：'old'，
newValue：'newValue'，
url：'url'，
storageArea：storage//更改的存储区域
}；
```

Webkit在开发者工具中提供了一个屏幕，程序员可以在这里查看和编辑localStorage和sessionStorage对象（见图4-1）。从开发者工具窗口，单击"Storage"选项卡。会显示一个页面中的localStorage和sessionStorage对象。存储屏幕也是完全可编辑的，可以在这里对键进行添加，删除和编辑。

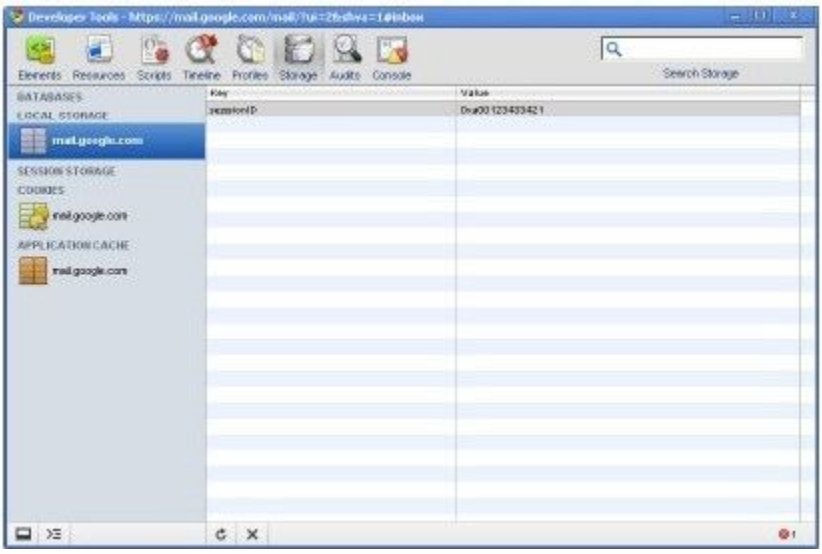

图　4-1　Chrome存储查看器

虽然Firebug不像Chrome和其他基于WebKit的浏览器那样提供local Storage和sessionStorage对象的接口，但是可以通过JavaScript控制台访问对象，而且可以在这里对键进行添加、编辑和删除操作。假以时日，希望有人会写一个Firebug扩展，实现这一功能。

当然，可以编写一个自定义界面查看和编辑任何浏览器上的存储对象。用例4-1中所示的GetItem和removeItem调用在屏幕上创建一个构件，并允许通过编辑文本框进行编辑。例4-3显示了一个构件的框架。

例4-3：存储查看器

```
(function createLocalStorageViewer()
{
$('<table></table>').attr(
{
"id": "LocalStorageViewer",
"class": 'hidden viewer'
```

```
    }).appendTo('body');
    localStorage.on('update', viewer.load);
    var viewer=
    {
    load: function loadData()
    {
    var data,buffer;
    var renderLine=function(line)
    {
    return"<tr key='{key}'value='{value}'>\n".populate(line)+
    "<td class='remove'>Remove Key</td>"+
    "<td class='storage-key'>{key}</td><td>{value}</td></tr
>".populate(line);
    };
    buffer=Object.keys(localStorage).map(function(key)
    var rec=
    {
    key: key,
    value: localStorage[data]
    };
    return rec;
    });
    };
    $("#LocalStorageViewer").html(buffer.map(renderLine).join(''));
    $("#LocalStorageViewer tr.remove").click(function()
    {
    var key=$(this).parent('tr').attr('key').remove();
    localStorage[key]=undefined;
    });
    $("#LocalStroageViewer tr").dblclick(function()
    {
    var key=$(this).attr('key');
    var value=$(this).attr('value');
    var newValue=prompt("Change Value of"+key+"?", value);
    if(newValue!==null)
    {
    localStorage[key]=newValue;
    }
    });
    };
    }());
```

在ExtJS中使用localStorage

ExtJS是一个非常流行的JavaScript框架，允许非常复杂的交互显示，在前面的章节中展示了一些ExtJS例子。本节说明如何使用ExtJS的localStorage。

ExtJS的一个不错的功能是许多对象可以记录状态。例如，ExtJS表格对象允许用户调整列的尺寸，隐藏和显示它们，并重新排列它们的顺序，当用户以后回到该应用时这些变化都记录并重新显示。这允许每个用户定制应用元素工作的方式的。

ExtJS提供了一个对象来保存状态，但要使用cookie储存数据。一个复杂的应用可以创造出足以超过Cookie大小限制的状态数量。一个包含几十个网格的应用可能超出cookie的大小，这样会锁定应用。因此使用localStorage会好得多，利用其较大容量的优势，避免了在每次请求中发送数据到服务器的开销。

建立一个自定义状态provider对象其实是很容易的。在例4-4中provider扩展了通用的provider对象和三个方法：set、clear和get。这些方法简单地读取和写入存储数据。在例4-4中，通过相当简单的方法使用"state_"字符串加上被保存元素的ID对存储的数据进行索引。这是一个合理的方法。

例4-4：ExtJS本地状态提供者Local State Provider

```
Ext.ux.LocalProvider=function(){
```

```
    Ext.ux.LocalProvider.superclass.constructor.call(this);
    };
    Ext.extend(Ext.ux.LocalProvider,Ext.state.Provider, {
    //*************************************************************
    set: function(name,value){
    if(typeof value=="undefined"||value===null){
    localStorage['state_'+name]=undefined;
    return;
    }
    else{
    localStorage['state_'+name]=this.encodeValue(value);
    }
    },
    //*************************************************************
    //private
    clear: function(name){
    localStorage['state_'+name]=undefined;
    },
    //*************************************************************
    get: function(name,defaultValue){
    return Ext.value(this.decodeValue(localStorage['state_'+name]),
defaultValue);
    }
    });
    //设置状态处理函数
    Ext.onReady(function setupState(){
    var provider=new Ext.ux.LocalProvider();
    Ext.state.Manager.setProvider(provider);
    });
```

也可以用一个大型对象存储所有状态数据，并将该对象存储到一个存储键中。这样做有一个优点，不会创建大量的存储元素，但会使代码比较复杂。此外，如果有两个窗口尝试更新存储，其中之一可能会干扰另一个窗口制造的变化。对于有竞争条件的问题，这里没有很好的解决方案。通常，在可能有问题的地方，使用IndexedDB或一些其他解决方案可能更好。

离线加载存储数据

当应用使用的某些持久性数据是相对静态的数据时，为更快速地访问，将其加载到本地存储中可能会有意义。在这种情况下，Ext.data.JsonStore对象需要进行修改，以便load()方法在localStorage区域中查找数据，然后再尝试从服务器加载数据。从localStorage加载数据后，Ext.data.JsonStore应该调用服务器来检查数据是否已经改变。这样，应用程序可以使数据立即对用户可用，而且只需花费尽可能小的短期开销。这样可以使用户界面感觉上更快，并减少应用程序占用的带宽量。

对于大多数的请求，数据不会改变，因此数据使用ETag的某种形式是非常有意义的。通过一个HTTP GET请求和一个If-None-Match头从服务器请求数据。如果服务器确定该数据并没有改变，它可以返回一个304 Not Modified响应。如果数据已更改，服务器发回新的数据，并在Ext.data.JsonStore对象和sessionStorage对象中都加载该应用。

Ext.data.PreloadStore对象（见例4-6）将数据作为一个大型的JSON对象（见例4-5）存储到session缓存中。进一步将服务器返回的数据包装到JSON闭包，这使得它能够存储一些元数据。在这种情况下，存储ETag数据和数据加载的日期。

例4-5：Ext.data.PreloadStore离线数据格式

```
{
"etag": "25f9e794323b453885f5181f1b624d0b",
"loadDate": "26-jan-2011",
"data": {
"root": [{
"code": "us",
"name": "United States"
},
{
"code": "ca",
"name": "Canada"
}]
}
}
```

注意：当构建一个ETag时，请务必使用一个好的哈希函数。MD5可能是最好的选择。也可以用SHA1，但它会产生一个非常长的字符串，因此可能不可取。从理论上讲，MD5可能会被破解，但在缓存控制的操作中，不需要为此担心。

localStorage对象中的数据可以从背景中改变。正如已解释过的，用户可以通过Chrome开发者工具或Firebug的命令行改变该数据。或者也可能只是碰巧用户有两个相同浏览器打开了同一个应用。因此监听localStorage对象的更新事件对store非常重要。

大部分的工作在beforeload事件处理函数中完成。该处理函数检查存储数据的缓存副本，如果存在，加载它到Store中。如果有数据出现，该处理函数也将重新载入数据，但会使用Function.defer()方法延迟

加载，直到系统有望完成加载网页的时间，因此这样做加载将很少会干扰用户。

store.doConditionalLoad()方法使Ajax调用服务器加载数据。它包含的If-None-Match头，这样，如果数据没有改变，就会加载当前的数据。它还包括一个强制选项，使beforeload处理函数加载新的数据，而不是试图刷新localStorage中对象缓存版本存储的数据。

一般把SECOND、MINUTE和HOUR定义为常数，这样做只是为了使代码更具可读性。

例4-6：Ext.data.PreloadStore

```
Ext.extend(Ext.data.PreloadStore,Exta.data.JsonStore，{
indexKey：''，
//默认索引键
loadDefer：Time.MINUTE，
//加载数据需要加载多长时间
listeners：{
load：function load(store,records,options)
{
var etag=this.reader.etag；
var jsonData=this.reader.jsonData；
var data=
{
etag：etag，
date：new date()，
data：jsonData
}；
sessionStorage[store.indexKey]=Ext.encode(data)；
}，
beforeload：function beforeLoad(store,options)
{
var data=sessionStorage[store.indexKey]；
if(data===undefined||options.force)
```

```
    {
    return true；//缓存中没有，从服务器加载
    }
    var raw=Ext.decode(data)；
    store.loadData(raw.data)；
    //推迟重新加载数据直到条件达到
    store.doConditionalLoad.defer(store.loadDefer,store，
[raw.etag])；
    return false；
    }
    }，
    doConditionalLoad：function doConditionalLoad(etag)
    {
    this.proxy.headers["If-None-Match"]=etag；
    this.load(
    {
    force：true
    })；
    }，
    forceLoad：function()
    {
    //在一个虚ETag中传递强制加载
    this.doConditionalLoad('')；
    }
    })；
```

为以后服务器同步存储变化

在一个应用可以离线使用或片状连接到互联网的事件中，向用户提供一种不需要有网络存在的保存修改的方法应该会获得不错的效果。要做到这一点，把每个记录中的变化写入一个localStorage对象的队列中。当浏览器再次联机时，队列可以推送到服务器。这类似于数据库中使用的事务日志。

存储队列如例4-7所示。队列中的每个记录代表在服务器上要采取的一个行动。当然，确切的格式将根据特定应用的需求而确定。

例4-7：存储队列数据

```
[
{
"url": "/index.php",
"params": {}
},
{
"url": "/index.php",
"params": {}
},
{
"url": "/index.php",
"params": {}
}
]
```

一旦Web浏览器回到联机状态，就有必要处理队列中的项目。例4-8获取该队列，并将第一个元素发送到服务器。如果该请求成功，则取下一个元素，并继续处理队列中的元素，直到整个队列发送完成。即使队列很长，因为Ajax以异步方式处理每个项目，所以该过程的执行也会实现对用户的影响最小。为了减少Ajax调用的数量，也可以将该代码修改为按组发送队列，每次发送5个。

例4-8：存储队列

```
var save=function save(queue)
{
if(queue.length>0)
{
$.ajax(
{
url：'save.php'，
data：queue.slice(0，5)，
success：function(data,status,request)
{
save(queue.slice(5))；
}
})；
}
}；
```

jQuery插件

如果所有客户端存储选项的不确定性多得让人抓狂，还可以有其他选择。因为JavaScript中有许多东西，可以用一个提供更好接口的模块覆盖一个不一致的坏接口。这里有两个这样的模块，可以使工作更轻松。

DSt

DSt（http://github.com/gamache/DSt）是一个包装localStorage对象的简单的类库。DSt可以是一个独立的或作为jQuery插件工作的类库。它会自动将复杂的对象转换成任何一个JSON结构。

DSt还可以保存和恢复表单的元素或者整个表单的状态。传递元素或元素的ID到DSt.store()方法可以保存和恢复元素。之后，要恢复该元素，则传递该元素到DSt.recall()方法。

要存储整个表单的状态，使用DSt.store_form()方法。它把ID或表单本身的元素作为参数。该数据可以用DSt.populate_form()方法恢复。例4-9是DSt最简单的使用方法。

例4-9：DSt接口

```
$.DSt.set('key', 'value');
var value=$.DSt.get('key');
$.DSt.store('element'); //存储一个form元素的值
$.DSt.recall('element'); //再次调用form元素的值
$.DSt.store_form('form');
$.DSt.populate_form('form');
```

---

jStore

如果不想分辨哪个浏览器支持什么存储引擎，也不想为每一种情况创建不同的代码，有一个好的解决方案：jQuery的jStore插件（http://twablet.com/docs.html?p=jstore）。该插件支持localStorage、sessionStorage、Gears SQLite和HTML 5 SQLite，以及Flash Storage（闪存）和IE7的专有解决方案。

jStore插件有一个简单的接口，允许程序员将“名称/值”对存储在不同的存储引擎中。它为所有引擎提供了一组接口，以便在给定浏览器不存在存储引擎的情况下，程序可以在需要时适当降级。

jStore插件在jQuery.jStore.Availability实例变量中提供了一个可用的引擎列表。程序应该选择最有意义的引擎。

对于需要多浏览器支持的应用，这可能是一个有益的补充。详情请参阅jStore网页。

第5章　IndexedDB

本地存储接口（见第4章）为存储少量数据提供了一个良好的接口，但是在许多情况下浏览器将此存储空间限制在5MB。如果应用的存储需要超过该限制，或者应用要求查询能够访问结构化的数据，那么本地存储就不是最好的选择。在这种情况下，应用开发人员有更健壮的存储机制可用。IndexedDB在许多浏览器上提供了这种机制。例如，程序员可以在IndexedDB中保存产品目录数据，因此，当用户搜索一个项目时，浏览器并不需要到服务器去寻找某个项目的数据。

IndexedDB是一种NoSQL数据库，如果使用过MongoDB或CouchDB这类产品，那么就会感到很熟悉。程序可以直接在IndexedDB数据存储中存储JavaScript数据。

Firefox从版本4开始包含IndexedDB。Google Chrome从版本11起也引入了IndexedDB。Microsoft在其HTML 5实验室网站（http://html5labs.interoperabilitybridges.com）有一个作为ActiveX控件的版本，可手动加入IE中，在可预见的未来可能出现在IE的主发布版本中，也可能在未来的一年或两年后，其他浏览器中也有IndexedDB可用。形式上，IndexedDB现在是一个来自W3C的建议草案（http://dvcs.w3.org/hg/IndexedDB/raw-file/tip/Overview.html）。

注意：因为到目前为止只有两个浏览器支持IndexedDB，所以IndexedDB的使用应限制在应用程序内部，可以限制所选择的浏览器。在一般的应用中使用它，应格外小心，Microsoft Internet Explorer、Opera和Safari还不支持此功能（至少到2011年7月为止）。

与localStorage一样，IndexedDB有着严格的同源策略。因此页面创建的数据库不能被其他主机上的网页访问。这为数据提供一个安全等级，因此它被保护不受其他页面上运行的脚本干扰。但页面上运行的任何创建数据库的脚本具有完全访问该数据库的权限，当然，用户具有该数据的访问权限。

IndexedDB比SQLite具有如下几个优势。首先，其原生的数据存储格式是一个JavaScript对象，不需要将一个JavaScript对象映射到一个SQL表结构，而SQL表结构不太好，易遭到SQL注入攻击。注入式攻击不会发生在IndexedDB上，虽然在某些情况下XSS可能是一个问题，例如用户将JavaScript加到数据库中，然后再放到一个页面中。

IndexedDB数据存储提供了一组接口，在本地磁盘上存储JavaSript对象。每个对象都必须有一个键，能通过该键检索对象，也可能还有第二个键。

与许多原生JavaScript接口一样，IndexedDB的原生接口也是非常晦涩，所以有点难用。但是又与许多其他JavaScript接口一样，

IndexedDB被封装成jQuery插件，因此API非常优雅。本章所有的例子都会用封装好的插件进行演示，因为这样的代码比使用原始接口的代码更容易理解。当然，如果可以选择的话我也建议你这样使用IndexedDB。

我将演示的例子都是围绕理论图书搜索的应用程序。每本书都会有一个记录格式，如例5-1所示。本例将按title和price字段进行索引，以便用户进行查询。

例5-1：设置事务版本

```
{
"title": "Real World Haskell",
"price": 49.95,
"price_can": 49.95,
"authors": [
"Bryan O'Sullivan",
"John Goerzen",
"Don Stewart"
],
"cover_animal": "Rhinosorus Beatle",
"cover_url": "http://...",
"topics": ["Haskell"]
}
```

IndexedDB通过内置的版本控制系统保持应用程序和相关的数据存储同步更新。每个数据存储都有一个版本，应用可以在加载时对其进行检查。如果不是最新版本，应用可以采取适当的动作，通过创建新对象和索引使其成为最新版本。这一部分已经自动化了，因为当结构有变化的时候，Indexed DB的jQuery接口会按需更新版本。

当加入了新的数据存储或索引时，版本应随之改变。当索引或存储被删除时，版本也必须相应改变。IndexedDB的jQuery模块会自动完成这些工作。我们只需要让应用程序检查不同的objectStores对象和indexes是否存在就能保证格式是正确的。如例5-2所示，如果index不存在就会自动创建一个。

例5-2：创建索引

```
$.indexeddb(db).objectStore(objectStore).index(field);
```

与IndexedDB的交互必须通过事务的方式实现。因为theIndexedDB接口是异步的，而且IndexedDB可以从一个Web Worker或运行在另一个线程中的第二个窗口访问（在JavaScript中的每个窗口中运行其自己的线程）。尽管同时访问的情况很罕见，但是多个JavaScript线程同时访问一个给定的IndexedDB数据库的情况还是可能出现，因此接口必须防止竞争条件。

注意：IndexedDB接口规范包括一个可以在Web Worker中使用接口的同步版本，但是还没有在浏览器中实现。此外，使用该接口的代码不能在WebWorker和非WebWorker环境之间重用。

IndexedDB中数据存储的基本单位是数据库。IndexedDB中的数据库是关系型数据库，与MySQL数据库的大致相同。每个IndexedDB数

据库包含一个或多个ObjectStores，可等同于SQL数据库中的表。但是与SQL数据库不同，ObjectStores没有固定的结构。

ObjectStores中的每个记录是一个键/值对，其中键是主索引，值是一个JavaScript对象。任何可序列化的JavaScript对象都可插入到存储对象中。

警告：IndexedDB中一般不能存储闭包和函数。

使用IndexedDB的第一步是创建一个新的数据库。这一步只需要选一个数据库名，并请求打开它。如果数据库不存在，就创建它。如果存在，则直接打开它。IndexedDB的jQuery插件让它非常简单，又需要给数据库一个名字就行：

```
$.indexeddb(db);
```

数据库打开后，要创建一个事务对象。创建该对象的函数将使用一个存储对象列表，以及一个可选的模式变量和超时。该模式变量可以是IDB Transcation.READ_ONLY、IDBTranscation.READ_WRITE或IDBTranscation.VERSION_CHANGE，默认为IDBTranscation.READ_ONLY。超时指定以ms为单位，如果没有超时，则可以设置为0。

由于IndexedDB的jQuery API为每个简单操作都隐式地创建了事务，所以只有在需要同时更新两个以上的objectStore或者做一些奇怪的操作时，才需要像例5-3一样创建显式事务。在使用map、forEach或者之类的操作循环添加列表项的时候，为了让所有数据存储时表现出一致行为，也有可能需要创建显式事务。当事务创建完成之后，需要调用transaction.done()提交事务，或者用transaction.abort()进行回滚。

例5-3：使用显式事务

```
    var transaction=$.indexeddb(db).transaction([],
    IDBTransaction.READ_WRITE);
    transaction.then(write,writeError);
    data.map(function(line){
    transaction.objectStore(objectStore).add(line).then(write,writeE
rror);
    return line;
    });
    transaction.done();
```

添加、更新记录

所有添加到一个IndexedDB数据库的数据必须要在一个事务的内部完成，这可以防止其他JavaScript进程修改同一数据。尽管JavaScript是单线程的，从一个Web Worker或者同一个浏览器中的第二个窗口中都可以打开该数据存储，并且都能同时修改同一数据。

一般来说，如例5-4所示，向objectStore对象中添加一条记录，只需要调用对象的add()方法。它会自动创建对应的事务。

例5-4：添加一行数据

```
   $.indexeddb(db).objectStore(objectStore).add(book).then(wrap,err
);
```

如果需要添加多条数据，可以使用例5-5所示的事务。在所有操作完成之后该事务才会被提交。调用transaction.done()方法之后会将事务会标记为完成，此时整个事务才会被提交。

例5-5：添加多行数据

```
   var transaction=$.indexeddb(db).transaction([],
IDBTransaction.READ_WRITE);
   transaction.then(write,writeError);
   data.map(function(record){
   transaction.objectStore(objectStore).add(record).then(write,writ
eError);
   });
```

正如add()方法一样，可以用update()方法更新一条记录。区别是，如果记录不存在那么update()方法会返回失败。

如果需要一次性更新多条记录，可以用cursor（详见本章的“检索数据”一节）遍历这组数据。这种情况下，用一个回调函数调用cursor的updateEach()方法。这个函数会被cursor指向的每个元素上调用，它

的返回值是该条记录更新后的值，这些值会在所有条目处理完成之后写回数据库中。返回值为false的行不会保存。

添加索引

索引只能在一个setVersion事务中添加或删除。索引可以在objectStore被创建的时候一起创建，也可以在objectStore创建之后调用index()方法完成（见例5-6）。index()方法在索引不存在的时候会创建索引，它还会返回一个index对象用于遍历该索引。

例5-6：使用索引

```
   $.indexeddb(db).objectStore(objectStore).index(field).openCursor
([1，10]).each(write)；
```

在IndexedDB类似的数据存储与多数其他数据存储之间的一个关键的区别是IndexedDB运行在客户的浏览器上，因此更新它需要比在中央位置的一小组服务器上运行的数据存储机制考虑得更多。JavaScript代码必须能够应付，用户可能有一个过期的存储及动态更新的可能性。

你可以检查用户电脑中存储对象正在使用的数据库版本，如果需要的话，还可以把版本更新到最新。举个例子，1.0版本的数据库软件创建了两个存储对象，在1.1版本又添加了第三个存储对象，并对存储对象中的一个数据库表添加了索引。通过代码检查存储对象的版本，可以知道是否需要对存储对象进行必要的更新，或者至少确认该用户

的存储对象是不是正常的。IndexedDB的jQuery插件会自动处理这些问题。

检索数据

在将数据存储到存储对象中后，需要有一种方法查询和显示数据。查询数据的方式有好几种：程序可能希望输出所有数据、一部分数据或者一条数据。

如例5-7所示，可以用get()方法通过主键检索存储对象中的一条数据。get()方法返回一条可以被then()方法访问的对象。then()方法需要两个回调函数作为参数：第一个参数在get()成功返回可访问对象时被调用，第二个参数是为以防万一返回错误对象时调用。

例5-7：得到一行数据

```
$.indexeddb(db).objectStore(objectStore).get(primaryKey).then(wrap,err);
```

通过索引查询数据库的单条或者特定范围的多条数据也很容易。首先指定一个特定的索引，然后用openCursor()得到cursor,openCursor()的参数需要指定一个可访问的范围，这个范围的两端可以是开区间也可以是闭区间。例5-8展示了一个典型的查询语句。

例5-8：得到一定范围内的数据行

```
$.indexeddb(db).objectStore(objectStore).index('title').openCursor([MIN,MAX]).each(write);
```

range是通过形式为[lower,upper,lowerExclusive,upperExclusive]的4元素数组设定的。前两个元素是整型，后两个元素是可选的，为布尔型。range默认是开区间的，也就是说[10，20]指定的范围是11～19。但是[10，20，false,false]指定的范围在搜索时会包含两端点。

如果只想用指定的上界或者下界搜索，只需要设置不用的参数为undefined。如[10，underfined,false]返回所有大于等于10的键值，而[undefined，10，undefined,true]返回所有小于10的键值。要得到一个特定的值，可以指定类似[10，10，false,false]这样的范围。

最后一种情况是需要得到整个存储对象的内容。这种情况下不需要指定索引（除非需要将记录排序），所以可以直接在objectStore对象上调用openCursor()。回调函数write会依次在每个元素上调用，如例5-9所示。

例5-9：得到所有行

```
   $.indexeddb(db).objectStore(objectStore).openCursor().each(write
);
```

删除数据

从一个存储对象中删除数据也很容易。每个存储对象有一个delete()方法，可以用将要删除的记录索引为参数进行调用。存储将删除这些记录，然后调用回调，如例5-10所示。

例5-10：删除数据

```
    $.indexeddb(db).objectStore(objectStore).remove(id).then(write,err);
```

你可以调用cursor的deleteEach()方法遍历和删除选中的行。这个方法的行为类似于数组过滤函数，删除函数中返回true的记录。如例5-11所示，删除函数isDuplicate()返回true的所有记录。

例5-11：删除重复的数据

```
    $.indexeddb(db).objectStore(objectStore).openCursor().deleteEach(function(value,key)
    {
    return isDuplicate(value);
    });
```

第6章　文件

由于众所周知的原因，浏览器访问文件系统的能力历来非常有限。虽然HTML表单已经能够上传文件，而且某些HTTP标头还可以让用户从服务器上下载文件。但是，除了这些特殊功能之外，浏览器不能够访问本地文件系统上的文件。一般来说，这是一件好事。谁都不希望访问的每个网页都能查看自己的硬盘！

HTML 5的一些新功能让浏览器访问文件系统有点受限。新的浏览器允许JavaScript通过现有表单中的文件输入域对文件进行访问。过去浏览器可以通过表单上传文件，而现在可以在JavaScript中通过文件直接使用数据。此外，现在浏览器还可以将文件从桌面拖动到Web应用中。Google的Gmail使用此功能允许用户添加附件。这一功能早先已在Flash中实现了，但是现在只需使用JavaScript就能实现了。

这样不会产生任何新的安全问题，因为应用程序先把这些数据上传到服务器，然后再下载到浏览器进行访问。

截至本书编写时，Firefox、Chrome和Opera已经支持这些功能。对于Safari和Internet Explorer，可以通过Flash插件实现文件拖曳。

二进制大对象

JavaScript对字符串和数字一直处理得很好，二进制数据从来就不是它的强项。但是最近JavaScript已增加了一个blob数据类型和接口，用于处理二进制大对象（blob）。JavaScript将blob视为一大块元数据。因此，JavaScript可以对blob进行的实际操作十分有限。然而，blob对移动二进制数据非常重要。blob可以从文件中读取或者写入文件中（本章后面的“文件处理”），作为URL的使用，存入IndexedDB（见第5章），传递给Web Worker或者从Web Worker接收（见第8章）。

可以用BlobBuilder接口创建一个新的blob。这样将创建一个基本的空BlobBuilder，可以从ArrayBuffer给它追加一个字符串、已有的blob或者二进制数据。BlobBuilder.getBlob()方法返回实际的blob。

BlobBuilder在Firefox 6中称为MozBlobBuilder，在Safari Nightly版本和Chrome 8中称为WebKitBlobBuilder。

例6-1中，原始PNG数据传递给一个BlobBuilder对象，用于创建一个blob，然后将blob变成一个URL。URL对象可以在DOM中通过一个图像标记的src属性进行设置。

例6-1：用原始数据创建一个blob URL

```
/*global window, $, BlobBuilder,document,XMLHttpRequest*/
/*jslint onevar: false,white: false*/
function makePNG(pngData)
```

```
{
var BlobBuilder=window.BlobBuilder||window.MozBlobBuilder||
window.WebKitBlobBuilder;
var blob=new BlobBuilder();
blob.append(pngData);
var url=blob.getBlob('image/png').createObjectURL(); return url;
}
```

利用slice()方法提取blob中的一部分。因为slice的参数与array和string的同名方法的参数不同，在Firefox和Chrome中已经分别被重命名为mozSlice()和webkitSlice()。无论浏览器请求什么名称，该方法都会将提取的数据作为新的blob返回。Slice是blob API允许对blob原始数据进行的唯一的访问。

如果一个blob包含需要加载的数据，即使是一个网址，它可以经过变换，通过createObjectURL()方法作为URL使用。这将返回一个可以分配给HTML标记属性（该标记属性需要一个URL）的对象。例如，一个包含图像数据的Blob可以被分配给一个＜img＞标记的src属性来显示图像。

当它是一个URL时，手动释放是非常重要的，因为JavaScript无法确定什么时候回收该对象。要做到这一点，使用revokeBlobURL()方法。Blob URL与创建脚本同源，因此它们可以在浏览器的同源策略可能有问题的地方灵活使用。当用户离开该页面时，浏览器也将撤销所有blob的URL。

操作文件

从很多年前开始，HTML表单就能够包含一个文件类型将其作为表单元素，并允许用户指定一个文件上传到服务器。浏览器不允许JavaScript设置这一表单域，因为担心不应该上传的文件可能会以某种方式被强制上传。然而，新的JavaScript API允许从已经被用户添加到表单元素中的本地系统阅读文件的内容。

包含文件上传域的表单将提供一个FileList对象。FileList对象的每个元素是一个File对象。文件对象向用户提供文件名称、MIME类型、大小和最后修改日期。完整路径不会暴露在JavaScript中，但是可以在Firebug中看到。

File对象指向的文件可以被FileReader对象完整读取或者使用.slice()方法部分读取。要上传一个非常大的文件，例如视频，需要将文件分割成较小的部分并分部上传到服务器，这种方法可能比上传整个文件（可能有几百兆）更好。还应当注意，在默认情况下，许多服务器将上传文件的大小限制在几十兆。

通过使用FileReader接口，程序可以读取该文件的内容。所有这些方法都返回void，并且浏览器通过onload处理函数将文件读入内存后交

付该数据。这个异步操作非常重要，因为将一个很大的文件读入浏览器可能要花一些时间。FileReader API有四个选项用于读取文件。

FileReader.readDataAsURL()

将文件转换到一个URL，使它可以在页面中使用，由此产生的对象将包含文件的完整数据。

FileReader.readAsText()

以字符串形式返回数据，默认情况下，返回的数据为UTF-8编码的文本，也可以指定不同的编码格式。

FileReader.readAsBinaryString()

以二进制字符串形式返回数据，不对内容进行解释。

FileReader.readAsArrayBuffer()

以一个ArrayBuffer形式返回数据。

例6-2展示了一个拖曳处理函数的例子。拖曳处理将在本章后面的“拖曳”部分完整介绍。对于本例需要明白的是，drop是一个传递给addEventListener的回调函数的名字，该回调函数由事件处理程序通过FileList对象表单中的文件列表进行调用。列表中的第一个文件动态创建一个URL，并将它添加到一个＜img＞标记显示在浏览器中。

例6-2：在文档中追加一个图片

```
var el=document.getElementById('dropzone')；
el.addEventListener('drop'，function(evt){
var file=evt.dataTransfer.file[0]；
file.readDataAsURL()；
file.onload=function(img)
{
$('div.images').append('＜img＞').attr({src：img})；
}；
})；
```

上传文件

把一个文件从桌面拖到浏览器中是个非常好的功能，但是如果不能把文件上传到服务器就会令人很郁闷，新版本中的XMLHttpRequest接口就提供了上传到服务器的功能。通过使用FormData接口，程序可以包装文件并将它们发送到服务器。XHMLHttpRequest还提供了一些回调，可以给用户提供反馈。onprogress事件返回已发送的字节数和上传的总大小，这可以在大量上传时显示进度条。上传完成时，onComplete事件被调用。

当使用XMLHttpRequest上传文件时，服务器会在与标准表单接口＜input type='file'multiple＞使用的相同接口中看到上传文件。因此，此服务器端代码应该是与文件上传相同的自表单元素被引入以来一直沿用的标准代码。

例6-3展示了用Ajax上传文件的示例代码。它使用formData对象包装要发送到服务器的文件。FormData是一个Java对象，用于把数据打包成可以被标准HTTP上传服务上传的文件。实现的方法是把文件名和文件内容当做一个blob传递给FormData.append()，然后用XHR2接口把FormData对象作为Ajax payload发送到服务器。

例6-3：用Ajax上传文件

```
function upload(files){
var uploadBlob=function uploadBlob(files,onload){
var xhr=new XMLHttpRequest();
xhr.open('POST', params.url,true);
xhr.onload=onload;
xhr.send(files);
};
var formData=new FormData();
files.map(function(file){
formData.append(file.fileName,file);
return file;
});
uploadBlob(files,function(){
alert("Upload Finished");
});
}
```

拖曳

HTML 5已经增加了对拖曳的支持。现在可以把一个或多个文件从桌面拖曳到浏览器，并用JavaScript访问这些文件。例如，它可以被文件管理器用来上传文件到服务器，或者被图形程序用来操作图像。一个社交网络站点可以允许用户拖曳图片到浏览器中进行裁剪、缩放、旋转以及在浏览器中预览，然后把它们上传到服务器。可以通过提供较小的图像节省服务器资源。

注意：为安全起见，文件API不会允许上传目录结构，而只是一个简单的文件清单。当然，如果正在构建一个文件管理器，可以让用户上传一个压缩文件（ZIP、RAR、tar等）并让服务器打开。有一些库可以用于多数流行的服务器端语言中解压大多数的压缩格式。

要实现拖曳，需要给目标DOM元素的drop事件添加一个监听。指定的事件处理函数将传递一个参数，其中有一个清单包含了被拖曳的所有文件。请注意，这个对象看起来像一个数组，因为它有数字键和一个长度属性。然而，它实际上不是一个数组，试图对它调用map或任何其他的数组操作符都不起作用。

警告：Selenium不能从桌面拖动文件到浏览器，因此也没有办法用Selneium自动测试拖曳。

全部整合到一起

为了演示本章介绍的特性如何协同工作以实现一个方便的文件处理接口。例6-4把这些全部放入一个更完整的示例中，这将Web页面上的一个区域设计为拖曳区。当用户把一个文件放到拖曳区时，它将被打包到FormData对象，然后通过XMLHttpRequest对象上传。这个函数还显示了一个进度指示器，将上传的进度反馈给用户。

例6-4：上传文件

```
/*global$，FormData,alert,document,XMLHttpRequest*/
/*jslint onevar：false,white：false*/
(function(){
var toArray=function toArray(files){
var output=[]；
for(var i=0，f；f=files[i]；i+=1){
output.push(f)；
}
return output；
}；
var updateProgress=function(state){
var progress=$('progress#progress')；
progress.attr('max'，100)；
if(state==='start'){
return progress.fadeIn(500)；
}
else if(state==='stop'){
return progress.fadeOut(500)；
}
//使用jQuery UI进度条
return progress.attr('value'，state)；
}；
var uploadBlob=function uploadBlob(params){
var xhr=new XMLHttpRequest()；
xhr.open('POST'，params.url,true)；
```

```
xhr.onload=params.success;
//监听上传进度。
xhr.upload.onprogress=params.progress;
xhr.send(params.blob);
return params;
};
var fileupload=function fileupload(){
var el=document.getElementById('dropzone');
var stopEvent=function(evt){
evt.stopPropagation();
evt.preventDefault();
};
el.addEventListener('dragover', stopEvent,false);
el.addEventListener('drop', function(evt){
stopEvent(evt);
var files=toArray(evt.dataTransfer.files);
updateProgress('start');
var packageFiles=function(files){
var formData=new FormData();
files.map(function(file){
formData.append(file.fileName,file);
return file;
});
var block={
url: '/upload.php',
success: function(){
updateProgress('done');
},
progress: function(evt){
updateProgress(100*(evt.loaded/evt.total));
},
blob: formData
};
return block;
};
uploadBlob(packageFiles(files));
}, false);
};
//运行设置
fileupload();
}());
```

首先创建函数toArray，它把所有文件名打包成一个数组。紧接着的updateProgess函数使用一个HTML 5提供的新进度条，这将在第10章

的“应用标记”一节介绍。uploadBlob函数代码与例6-3的代码一样，fileupload函数也使用例6-3的代码，加上了一些例6-2的代码。

Filesystem文件系统

浏览器允许JavaScript访问文件系统的想法足以让任何考虑到安全问题的人陷入恐慌。用户的硬盘驱动器上有很多不希望浏览器访问的东西。

GoogleChrome允许JavaScript访问用户计算机上的沙箱文件系统。如果想从网页内运行FileSystem API,Chrome浏览器必须用--unlimited-quota-for-files标记启动。不过，如果正在为Google的网上商店建立一个应用，那么可以通过在存储清单文件中指定unlimitedStorage访问这个API。

虽然LocalStorage和IndexedDB允许JavaScript程序在数据库中存储对象，但是FileSystem API对于存储大型二进制对象非常有用。例如，如果正在构建一个视频应用，在文件系统上处理图像可能会对存储很有用。其他用例可以包括视频流、具有很多媒体资源的游戏、图像编辑或音频应用。总之，这个接口将适合于任何需要在本地短期或长期存储大量数据的应用。

注意：由于文件系统API只在Chrome浏览器中支持，而且只有当用户加载一个受信任的应用时才支持，这有点超出了本书的范围。在

《Using the HTML 5 Filesystem API》一书中可以找到完整的相关知识。

第7章　离线处理

近来人们使用互联网似乎一直很通畅，但是，坦率地说不是这样。在有些时候和有些地方，即使最先进的移动设备也会出于这样或那样的原因离开网络的覆盖范围。

第4章着眼于如何将本地存储的数据传递给浏览器，使浏览器不需要访问网络即可使用。然而，如果没有托管应用的Web页面，再方便的数据也是没有用的。

随着越来越多现代的应用进入到浏览器，能够随时访问该软件已经变得至关重要。问题是，标准的Web应用假定Web页面将要加载许多组件，包括JavaScript源代码、HTML、图像、CSS等。为了用户能够在没有接入互联网时利用这些资源，需要在本地存储这些文件的副本，在需要的时候供浏览器使用。HTML 5让程序员能够给浏览器提供一个要加载和保存的文件清单。即使是没有网络连接到服务器，浏览器也将能够访问这些文件。

即使浏览器在线，清单中列出的文件也将从本地磁盘加载。从而使终端用户体验到终极内容交付网络。

当页面加载时，只要浏览器在线就会检查服务器的清单文件。如果清单文件已更改，浏览器将尝试重新下载清单中列出的所有需要下

载的文件。一旦下载完清单中的所有文件，浏览器将更新文件缓存以显示新的文件。

清单文件简介

文件离线访问是由Google在Gears中推出的一项功能。用户提供了一个JSON文件形式的清单，可以引导浏览器加载其他离线请求的文件。当浏览器下次访问该网页时，文件将从本地磁盘加载，而不是从网络加载。当清单文件的版本字段被更新时，Gears会检查清单中所有文件进行更新。

HTML 5清单的想法与上面所说的类似，但在实现上有所不同。它的一个好处是，不用任何JavaScript代码就可以在应用中实现一个清单，但是Gears中需要用JavaScript代码实现。要做到这一点，需要在文档的＜html＞标记（见例7-1）中增加manifest属性来包含清单文件的路径。

例7-1：HTML manifest声明

```
＜!DOCTYPE HTML＞
＜html manifest="/cache.manifest"＞
＜body＞
...
＜/body＞
＜/html＞
```

清单文件的MIME类型必须为text/cache-manifest。可以通过Web服务器配置文件来实现。设置Apache Web服务器的MIME类型，在Apache的配置文件中添加下面这行代码。对于其他Web服务器，请参考相关服务器的文档。文件的名称并不重要，只要文件有正确的MIME类型，但是cache.manifest似乎是一个很好的默认选择。

```
AddType text/cache-manifest.manifest
```

清单文件的结构

清单文件的格式其实非常简单。第一行必须是"CACHE MANIFEST."。之后是一个文件列表，每行一个，包含在清单中（见例7-2）。注释可以用井号（#）标注。

该清单将缓存HTTP GET请求，而POST、PUT和DELETE仍需访问网络。如果页面有一个活动的清单文件，所有的GET请求都将被定向到本地存储。但对于某些文件，离线访问没有意义。这些可能包括各种服务器资源，例如Ajax调用或者可能会溢出缓存区的大型文档集。这些文件可以包含在清单中的NETWORK部分。NETWORK部分的所有URL都将绕过缓存直接从服务器加载。HTML 5的清单要求任何清单中未包含的文件选择清单以外的办法。[1]

在其他情况下，可能希望根据用户是否在线提供不同的内容。清单为每个资源提供了一个FALLBACK部分。将根据浏览器是否连接到互联网，向用户显示不同的内容。FALLBACK部分的每一行的第一个文件用于当连接可用时从服务器加载，第二个文件用于当连接不可用时从本地加载。

NETWORK和FALLBACK部分都是列出文件模式，而不是具体的文件。因此可以在这里列出整个目录或URL路径以及文件类型，如图

像（*.jpg）。

例7-2：清单文件

```
CACHE MANIFEST
#2010-10-11
/index.php
/js/jquery.js
/css/style.css
/images/logo.png
NETWORK:
/request.php
FALLBACK:
/about.html/offline-about.html
```

[1] HTML 5中，对于清单中未包含的文件，当用户离线时不加载，在线时从服务器加载。

更新清单

清单文件本身一经改变，浏览器就会更新清单中的文件。有几种方式用来处理这个问题。可以在文件的注释中增加一个版本号。如果项目使用了一个与Subversion类似的版本控制，则可以使用版本号标记来解决这个问题。

用版本控制系统中的版本号有一个问题，每当系统中的任意文件发生变化时，程序员都得记着更新文件。创建一个自动系统，并作为部署过程的一部分运行该脚本要好得多。只要清单中列出的文件有变化就会自动更新清单文件。

例如，可以写一个脚本，检查清单中所有文件的变化，然后当其中一个文件变化时对清单文件本身进行修改。一个简单的方法是写一个脚本，对清单中的所有文件进行循环，然后对每一个做MD5校验，再将最后的校验放入manifest文件。这将确保任何更改都会引发清单文件更新。

该脚本可能会太慢而不能从Web服务器运行，因为要在1s内运行数百次很困难。但是，它可以有效地运行在开发环境中，可以选择当文件保存时从编辑器中运行，或者作为版本控制系统中检查过程的一部分运行。

在例7-3中，对清单文件进行解析，并用它做一些事情。该程序用Symfony Yaml Library（http://components.symfony-project.org/yaml）类库加载文件列表作为清单使用。另外，该程序首先检查有没有文件重复加入。还会检查每一个文件是否存在，缺少文件会破坏清单。脚本通过用注释在文件名后面添加每个文件的MD5，确保任何文件的更新都会引发清单文件的变化，以便浏览器更新其内容。数据文件格式如例7-5所示。例7-3将输出一个清单文件，文件中包含一个MD5 Hash作为注释，见例7-4。

例7-3：自动更新清单文件

```
＜?php
header('Content-Type：text/cache-manifest')；
echo("CACHE MANIFEST\n")；
$files=sfYaml：load('manifest.yml')；
$hashes=''；
$files=unique($files)；
foreach($files-＞cache as$file)
{
if(file_exists($file))
{
echo$file."\n"；
$hashes.=md5_file($file)；
}
}
echo"\nNETWORK：\n"
foreach($files-＞network as$file)
{
echo$file."\n"；
}
echo"\nFALLBACK：\n"
foreach($files-＞fallback as$file)
{
echo$file."\n"；
}
echo"#HASH：".md5($hashes)."\n"；
```

例7-4：包括MD5 Hash的清单

```
CACHE MANIFEST
index.html
css/style.js
js/jquery.js
js/myscript.js
NETWORK:
network/file
FALLBACK:
/avatars//offline-avatars/offline.png
#HASH: 090c7e8fe42c16777fba844f835e839b
```

例7-5：例7-3的数据

```
files:
-index.html
-css/style.css
-js/jquery.js
-js/myscript.js
network:
-network/file
faillback:
-/avatars//offline-avatars/offline.png
```

警告：当以为清单应该非常好地更新时候，它不一定总是很好。即使有一个清单的新版本，可以经常花一些时间来更新浏览器中的内容。除非设置缓存控制头，否则浏览器要在最后下载完几个小时后才会再次下载清单。例如，确保缓存控制头不会导致浏览器每隔五年才下载文件或使用ETag头，或者将服务器设置为没有缓存头，一定要测试好。

事件

浏览器通过清单文件加载页面时，会在window.applicationCache对象上触发一个检查事件。此事件将检查该页面之前是否已被访问过。

如果缓存之前尚未下载，浏览器就会触发一个downloading事件，并开始下载文件。如果清单文件发生改变此事件也会触发。如果清单中没有变化，浏览器将触发noupdate事件。

当浏览器下载文件时，会引发一系列progress事件。如果想要向用户提供某种形式反馈，让用户知道软件正在下载，可以使用这些事件。

一旦下载完所有文件，就会触发cached事件。

如果出现任何错误，浏览器将触发error事件。这可能是由有问题的HTML页面、有缺陷的清单、下载失败或者清单中列出的任何资源下载失败造成的。如果清单中一个文件缺少，则清单中的任何文件都不会下载。当清单正在改变的情况下，如果有一个坏连接，该文件的旧版本将被保留。在新清单中浏览器可能直接忽略该清单。然而，可能不是所有的浏览器或浏览器版本都在这一点上保持一致。用自动测试验证清单中的所有URL是一个好主意。对缓存来说，这可能是一个非常严重的错误，因为几乎没有可见的证据证明什么地方出了错。捕

捉错误对象并呈现给用户，作为对坏链接自动测试的某种形式是不错的想法。

在GoogleChrome浏览器中，开发人员工具可以显示清单中的文件列表（见图7-1）。在"Storage"选项卡下的"Application Cache"子项将显示各项目的状态。

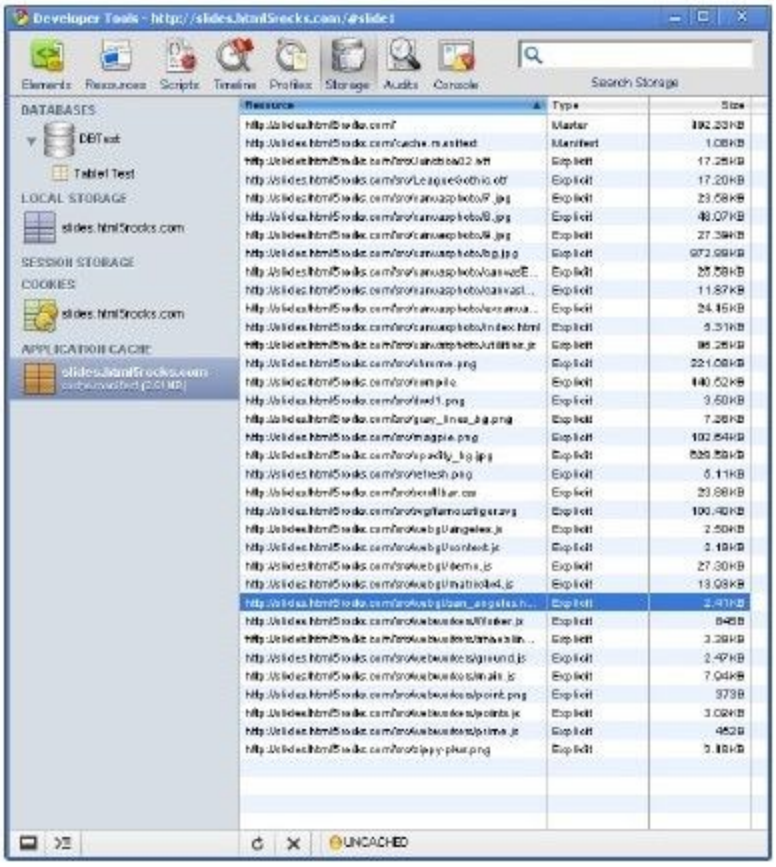

图　7-1　Chrome存储查看器

警告：在开发过程中关闭清单文件，只有当项目准备使用时才启用，这是个不错的想法。如果变化不能快速出现，使用缓存会使应用

非常难以开发。

调试清单文件

清单文件向特殊的调试提出了挑战。它可能是几种特殊类型错误的来源。

第一个最明显的错误是在清单中包括不存在的文件。如果一个文件被包含在页面中但是在清单中没有，它将不会被页面加载，同样，服务器上缺少的文件也不会被下载。

许多Selenium测试不会明确地测试正确的样式和显示的图片，因此很可能一个缺少CSS文件或图片的应用在某种程度上仍然可以工作，它通常是在Selenium中测试的。在一个包括来自外部Web服务器资源的应用中，这些也必须在清单文件的白名单中列出。

在一些浏览器中，包括Firefox，出现了进一步的问题，这些浏览器中清单具有了选择功能。因此Selenium测试可能不会选择它，这将使整个测试失去实际意义。为了在Firefox中测试，有必要建立一个应用缓存可以默认使用的Firefox配置文件。步骤如下：

1.完全退出Firefox。

2.从命令行用-profileManager切换启动Firefox。这样会出现一个如图7-2所示的对话框。保存自定义配置文件。

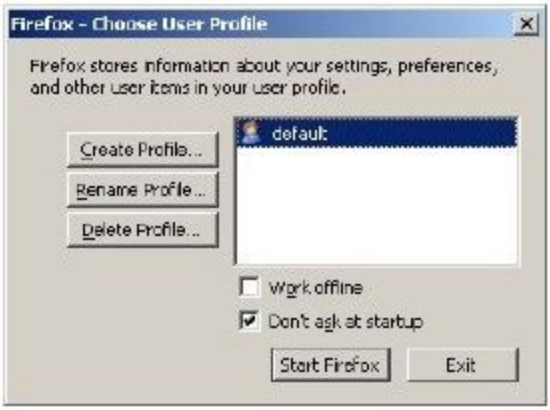

图　7-2　Firefox自定义配置对话框

3.重新启动Firefox。转到Firefox的选项菜单，选择"Advanced"选项卡，在下面选择"Network"选项卡（见图7-3），然后关闭“当一个网站要求存储数据离线使用时，告诉我”选项。

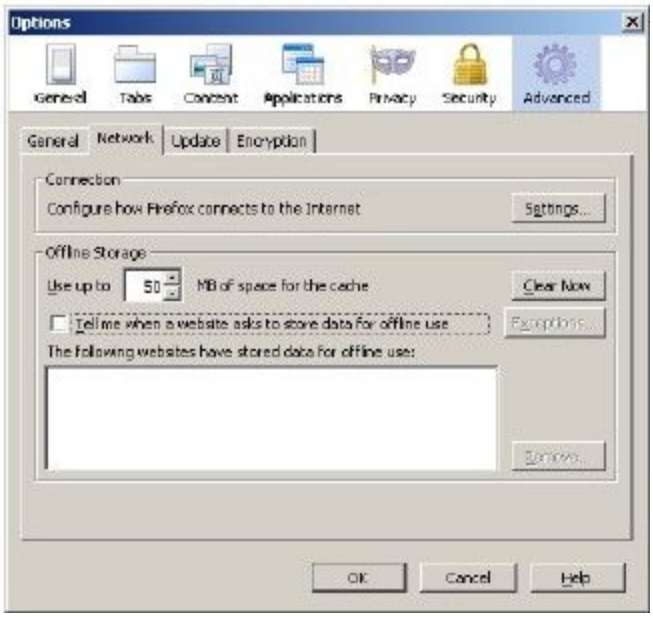

图　7-3　Firefox选项

现在，当启动Selenium RC服务器时，使用此选项：

```
java-jar selenium-server.jar-firefoxProfileTemplate
```

Firefox配置文件的完整细节参见网址：http://support.mozilla.com/en-US/kb/Managing+profiles。

当清单更新而浏览器没有反映该更新时，会出现第二类问题。通常情况下，会在加载页面后花费一两分钟的时间让浏览器更新文件缓存，并且直到页面加载完成浏览器才会检查缓存。因此，如果服务器被更新，浏览器在用户访问该页面之前将不会有新的版本。如果在服务器上已经有了一个更新，就会引起问题，会导致浏览器中的应用失败，例如Ajax协议的变化。

当用户访问该页面时（当然假设该浏览器在线），浏览器会从服务器上获取清单文件。但是，如果清单文件中设置了一个缓存控制头，浏览器可以不检查清单的新版本。例如，如果文件中有一个头，声明浏览器应该一年只检查一次更新（Web服务器上有时很常见），浏览器将不会重新载入清单文件。所以确保清单文件本身不被浏览器缓存是非常重要的，或者如果它被缓存了也只能通过一个ETag来实现。

浏览器可以通过给URL附加一个查询字符串cache.manifest?load=1以阻止缓存清单文件。如果清单文件是一个静态的文本文件，查询字

符串将被忽略，但浏览器不知道，浏览器将要求服务器发送一个新的副本。

不同的Web浏览器甚至一个浏览器的不同版本对清单文件的更新都会有所不同。因此在使用一个清单文件时，非常仔细地对任意应用进行跨不同浏览器和浏览器版本测试是非常重要的。

第8章　把工作分割成Web Worker

JavaScript自有史以来一直以单线程运行。这对小应用是可行的，但现在随着应用越来越大，它变得很局限。随着运行的JavaScript越来越多，应用开始阻塞等待代码执行完成。

JavaScript从一个事件循环运行代码，该循环从浏览器中已发生的所有事件队列中提取事件。一旦JavaScript运行时闲置下来，就会取出队列中的第一个事件，并执行响应该事件的处理程序（见图8-1）。只要这些处理程序快速运行，就能制造出应答的用户体验。

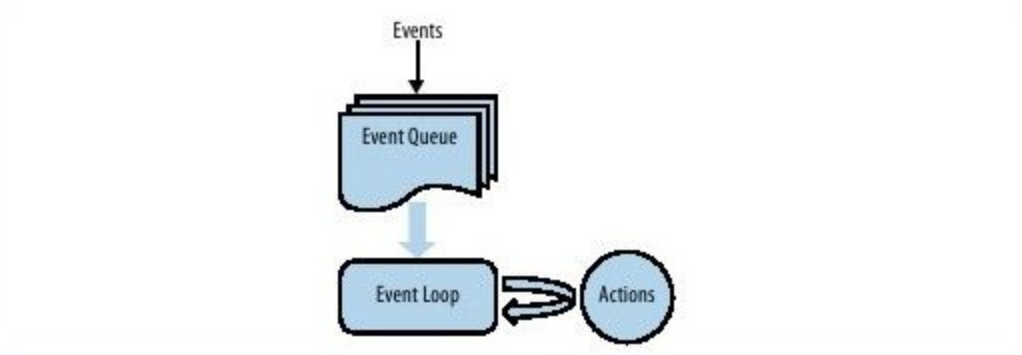

图　8-1　事件循环

在过去的几年中，浏览器之间的竞争一直围绕着JavaScript的速度。在Chrome和Firefox中，JavaScript现在的运行速度要比以前它在IE6中的速度快多达一百倍。因此，它可以在事件循环中加入更多代码。

值得庆幸的是，JavaScript做大多数事情的速度都很快。往往是按顺序操纵一些数据，并传递给DOM或进行Ajax调用。所以图8-1的模型工作得很好。对于那些需要花费比几分之一秒更长的时间计算的事情，有一些技巧可以防止计算瓶颈影响用户体验。

一个主要的技巧是把计算分割成多个小步骤，并在队列中将每个步骤作为独立的工作运行。每一步随着下一个步骤的调用而结束，其间进行短暂延迟（1/100s）。这可以防止任务锁定事件队列。但是因为它将任务调度的工作留给了程序员，而有效地实现这个解决方案也需要付出很大的努力，因此仍然从根本上不合格。

如果时间步长太短，计算可能仍会堵塞事件队列，并导致其他任务延后。虽然任务仍然会发生，但是因为系统没有马上回应点击和其他用户可见的行为，用户会感觉到滞后。另一方面，如果行动之间的步骤太大，计算将需要很长的时间来完成，导致用户需要等待结果返回。

GoogleGears创建了“Worker池”的思路，现在已经转化成HTML 5 Web Worker。接口有所不同，但基本思路相同。一个Worker是一个单独的JavaScript程序，可以执行计算、回传信息并往返于主程序和其他Worker之间。Web Worker不同于Java或Python中的线程，设计的一个关键方面是：没有共享的状态。Worker和主要的JavaScript实例只能通过传递消息进行通信。

这一区别导致一些关键的编程实践大多比线程编程简单。Web Worker不需要互斥、锁定或同步，不会出现死锁和竞争条件。这也意味着可以使用大量的JavaScript包，不用担心它们是否是线程安全的。浏览器的JavaScript环境唯一的变化是一些新的方法和事件。

每个Worker（包括主窗口）维持一个独立的事件循环。每当没有代码运行时，JavaScript运行时返回到事件循环。这时运行时将取出队列中的第一条消息。如果队列中没有事件，它会等待，直到一个事件到达，然后处理它。如果某些代码正在长时间运行，将一直到这段代码完成后才会处理事件。

这将在主窗口中导致浏览器的用户界面锁定。（有些浏览器提供将JavaScript停止在这一点上的支持。）在一个Worker中，一个较长的任务将阻止Worker接收任何新的事件。但主窗口和任何其他Worker将继续响应。

然而，这种设计的选择给Worker进程本身附加了一些约束。首先，Worker没有访问DOM的权限。这也意味着当Firebug与JavaScript通过DOM通信时，Worker不能使用Firebug的控制台界面。最后，JavaScript调试器不能访问Worker，因此没有办法逐步调试代码或者做那些调试器中通常能完成的其他事情。

Web Worker用例

在网络上运行的各类传统应用程序以及网页浏览器环境的限制，限制了调用Web Worker的计算需求。直到最近，大多数Web应用程序能操纵少量的数据，主要包括文字和数字。在这些情况下，有限地使用Web Worker的构造类型。现在，JavaScript被要求做更多的事情，许多常见的情况可以从繁衍的新任务中受益。

图像

HTML 5的＜svg＞和＜canvas＞的标记允许JavaScript操作图像，图像处理是一个潜在的繁重的计算任务。虽然自1993年Mosaic浏览器发布以来，Web浏览器已经能够显示图像，但是浏览器依旧无法操作这些图像。如果一个web程序员想取一个图像进行扭曲、透明叠加等，不能在浏览器中进行。在＜img＞标记中，所有的浏览器可以做的只有通过改变src属性替换一个不同的图像，或改变显示图像的大小。然而，浏览器没有办法知道图像是什么，或访问组成图像的元数据。

最近加入的＜canvas＞标记将一个已有的图像导入到画布上，并将元数据导出到JavaScript中处理，只要图片是从页面所在的同一台服务器上加载的，也可以从HTML 5＜video＞标记的一个视频中导出一帧。[1]

从图形中提取出数据，可以把它传递给一个Worker进行处理。这可用于做任何事情，从清理图像到对科学数据集做傅里叶变换。Canvas使得构建复杂图片，通过各种用JavaScript编写的过滤器编辑成为可能，应该经常使用Web Worker以获得更好的性能。

[1] HTML 5图像的更多信息参见《HTML 5 Canvas》，作者Steve Fulton和Jeff Fulton（O'Reilly）。

地图

除了图形以外，JavaScript现在还有可以用于处理地图数据的API。可以从互联网导入地图，并找出用户的当前位置，通过地理信息可以实现广泛的网络应用服务。

假设在移动浏览器中构建了一个路线查找应用。拿出电话，告诉它想要去“特拉维夫国王乔治大街14号”，然后让浏览器找出你所在的位置，指出到最近公共汽车站的路线，并告诉你应该从拉马特甘的钻石区乘82路公交车到达那里，应该是不错的体验。

该软件更复杂的版本可以检查交通状况，告诉你一个不同的公交车，虽然可能走绕远一点的路线，而且离目的还有一段距离需要步行，但可能会更快到达，因为错开了主要交通拥堵。

使用Web Worker

要启动一个Web Worker，创建一个新的Worker对象，并传递包含该代码的文件作为调用的参数（见例8-1）。下面的代码将从源文件创建一个Worker。

例8-1：Worker示例

```
$(document).ready(function(){
var worker=new Worker('worker.js');
worker.onmessage=function(event){
console.info(event);
};
worker.postMessage("World");
});
```

浏览器加载该Worker，运行所有不在事件处理程序中的代码，然后启动事件循环等待事件。要关注的主要事件是message事件，它定义了发送数据到worker的方法。主线程通过调用postMessage函数发送message事件，将传送的数据作为参数。

来自主线程的数据保存在event.data字段中。Worker应Message()方法读取这些数据。

Worker环境

Web Worker运行在一个非常小的环境中。缺少许多浏览器中熟悉的对象和JavaScript接口，包括DOM、文档对象、窗口对象。

除了标准ECMA脚本对象外，如字String、Array和Date，还有下列对象和接口对Web Worker可用：

·navigator对象包含四个属性：appName、appVersion、userAgent和platform。

·location对象的所有属性为只读。

·self对象就是Worker对象本身。

·importScripts()方法。

·XMLHttpRequest接口用于Ajax方法。

·setTimeout()和setInterval()。

·close()方法用于结束worker进程。

也可以使用ECMAScript5 JSON接口，因为它们是语言的一部分而不是浏览器的一部分。此外，Worker可以用importScripts()方法从服务器导入库脚本。该方法的参数为要加载的一个或多个文件列表，与主用户界面线程中使用的＜script＞标记具有相同的效果。与JavaScript中的大多数方法不同，importScripts是阻塞式的，直到列出的所有脚本加

载完成后该函数才会返回。importScripts将根据指定的命令顺序执行加载的文件。

虽然localStorage和sessionStorage不能从Web Worker访问，但是IndexedDB数据库可以（见第5章）。此外，IndexedDB规范规定可以在Web Worker中（而不是在主窗口中）使用阻塞调用表单。因此，在Worker使用IndexedDB操作数据的情况下，把新的数据加载到数据库中，然后发送一个"updated"消息到主窗口或其他Worker，让它们知道以便采取任何必要的行动。

Worker通信

Worker相关的主要事件是message事件，从主要JavaScript环境中的postMessage方法向Worker发送传递信息。在Firefox中，可以传递复杂的JavaScript对象。然而，有些版本的Chrome和Safari只支持简单的数据，如字符串、布尔和数字。一个很好的思路是将所有数据编码成JSON，再发送到一个Web Worker。

Worker可以通过同一个postMessage方法将数据发送回主线程。通过worker.onmessage处理程序接收回主线程。

Worker的通信模型是创建Worker的主要任务，然后来回传递消息，如图8-2所示。

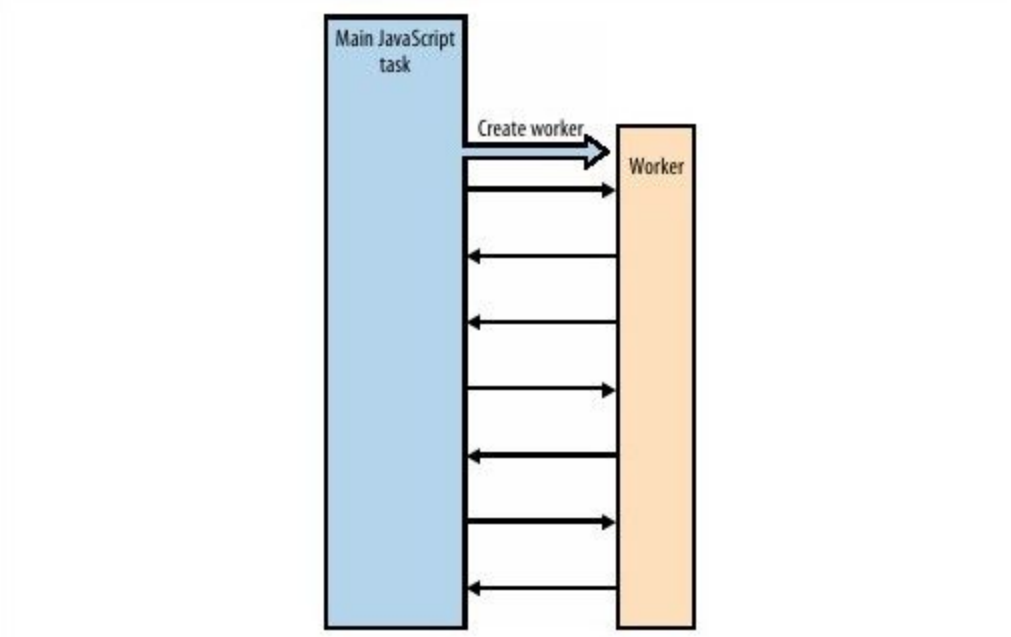

图　8-2　Worker通信

Web Worker碎形示例

例8-1是Web Worker的"Hello World"示例，它调用了一个更为复杂的例子。图8-3显示了一个Web Worker中计算的Mandelbrot集的视觉效果。这里，该Worker和主线程对工作进行了分割以绘制碎形。Worker完成了Mandelbrot集的实际计算工作，而前端脚本则获取原始数据，并显示在画布上。

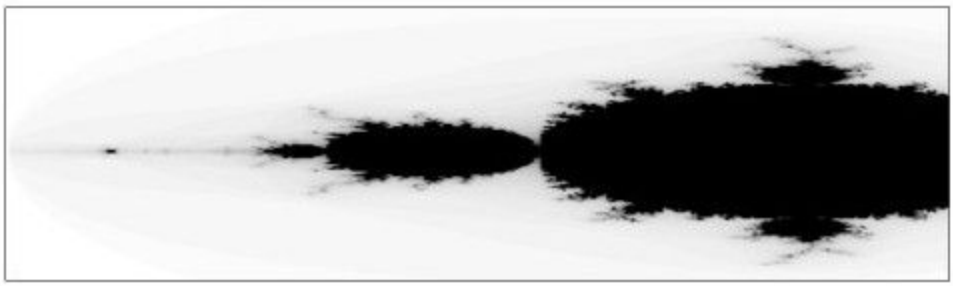

图　8-3　Mandelbrot示例

前端脚本（见例8-2）创建canvas元素，并进行缩放使其适合页面。然后，创建一个对象包装Worker接口。包装器对象在包装器的run()方法中创建Worker，传递给Worker一个参数块，告诉它要计算哪一块Mandelbrot集。

draw方法提取数据，进行缩放使它适合画布大小，设置一种颜色，然后绘制像素。

注意：HTML Canvas没有绘制像素的命令，因此要绘制一个像素，必须绘制一个大小为1的方块，并且从目标显示位置偏移半个像素。因此，要在（20，20）点处绘制一个像素，它应该是在（19.5，19.5）～（20.5，20.5）。在画布网格上的位置不是屏幕上的像素，而是在它们之间的点。

然后onMessage处理程序，等待要从Worker发送的事件。如果事件类型是draw，处理程序将调用该方法在画布上绘制新的数据。如果事件是log，通过console.info()记录到JavaScript控制台。为记录Worker的状态信息提供了一个非常简单的方法。

startWorker将this赋值给一个局部变量，别名为that。这是因为this不像其他的JavaScript变量受语法作用域限制。要让内部函数访问需要绘制一个像素的对象，有必要将它的别名命名为一个语法作用域内的变量。按照惯例，该变量通常被命名为that。

例8-2：Mandelbrot前端

```
var drawMandelSet=function drawMandelSet(){
var mandelPanel=$('body');
var width=mandelPanel.innerWidth();
var height=mandelPanel.innerHeight();
var range=[{
x: -2,
y: -1.4
}, {
x: 5,
y: 1.4
}];
```

```
$('canvas#fractal').height(height+100);
$('canvas#fractal').width(width-50);
var left=0;
var top=0;
var canvas=$("canvas#fractal")[0];
var ctx=canvas.getContext("2d");
var params={
range: range,
startx: 0.0,
starty: 0.0,
width: width,
height: height
};
var y_array=[];
var worker={
params: params,
draw: function draw(data){
data.forEach(function d(point){
if(this.axis.x[point.drawLoc.x]===undefined){
this.axis.x[point.drawLoc.x]=point.point.x;
}
if(this.axis.y[height-point.drawLoc.y]===undefined){
this.axis.y[height-point.drawLoc.y]=point.point.y;
}
ctx.fillStyle=pickColor(point.escapeValue);
ctx.fillRect(point.drawLoc.x+0.5,
height-point.drawLoc.y+0.5, 1, 1);
}, this);
},
axis: {
x: [],
y: [],
find: function(x,y){
return new Complex(this.x[x], this.y[y]);
},
reset: function(){
this.x=[], this.y=[];
}
},
myWorker: false,
run: function startWorker(params){
this.myWorker=new Worker("js/worker.js");
var that=this;
this.myWorker.postMessage(JSON.stringify(params));
this.myWorker.onmessage=function(event){
var data=JSON.parse(event.data);
if(data.type==='draw'){
that.draw(JSON.parse(data.data));
```

```
    }
    else
    if(event.data.type==='log'){
    console.info(event);
    }
    };
    }
    };
    worker.run(params);
    return worker;
    };
    $(document).ready(drawMandelSet);
    Function.prototype.createDelegate=function createDelegate(scope)
{
    var fn=this;
    return function(){
    fn.call(scope,arguments);
    };
    };
    function pickColor(escapeValue){
    if(escapeValue===Complex.prototype.max_iteration){
    return"black";
    }
    var tone=255-escapeValue*10;
    var colorCss="rgb({r}, {g}, {b})".populate({
    r: tone,
    g: tone,
    b: tone
    });
    return colorCss;
    }
    String.prototype.populate=function populate(params){
    var str=this.replace(/\{\w+\}/g,function stringFormatInner(word)
{
    return params[word.substr(1, word.length-2)];
    });
    return str;
    };
```

Worker本身实际很简单（见例8-3）。它只是加载了其他一些文件，然后等待一个从用户界面发来的消息。当收到一个消息时，开始计算。

例8-3：Mandelbrot计算

```
   importScripts('function.js', 'json2.js', 'complex.js', 'computeMa
ndelbrot.js',
   'buildMaster.js');
   onmessage=function(event){
   var data=typeof event.data==='string'?JSON.parse(event.data):
event.data;
   buildMaster(data);
   };
```

生成函数（见例8-4）对Mandelbrot集合中的点网格进行循环，计算每个点的转义值（见例8-5）。每计算200点后，生成函数发送其计算结果到主线程进行绘图，然后归零其计算点的内部缓冲区。此方式不是等待整个网格一次性绘制，用户能看到图像逐步生成。

例8-4：Mandelbrot生成

```
   var chunkSize=200;
   function buildMaster(data){
   var range=data.range;
   var width=data.width;
   var height=data.height;
   var startx=data.startx;
   var starty=data.starty;
   var dx=(range[1].x-range[0].x)/width;
   var dy=(range[1].y-range[0].y)/height;
   function send(line){
   var lineData=JSON.stringify(line.map(function
makeReturnData(point){
   return{
   drawLoc: point.drawLoc,
   point: point.point,
   escapeValue: point.point.mandelbrot()
   };
   }));
   var json=JSON.stringify({
```

```
type: 'draw',
data: lineData
});
postMessage(json);
};
function xIter(x,maxX,drawX){
var line=[];
var drawY=starty;
var y=range[0].y;
var maxY=range[1].y;
while(y<maxY){
if(line.length%chunkSize===chunkSize-1){
send(line);
line=[];
}
var pt={
point: new Complex(x,y),
drawLoc: {
x: drawX,
y: drawY
}
};
line.push(pt);
y+=dy;
drawY+=1;
}
send(line);
if(x<maxX&&drawX<width){
xIter.defer(1, this, [x+dx,maxX,drawX+1]);
}
}
xIter(range[0].x,range[1].x,startx);
}
```

此应用的最后一部分是Mandelbrot集合的实际数学计算，如例8-5所示。此功能用一个while循环实现而不是一个纯函数（见第2章的“函数式编程”），因为JavaScript不支持尾递归。用一个递归函数实现更酷，但将有可能导致堆栈溢出。

例8-5：Mandelbrot计算

```
Complex.prototype.max_iteration=255*2;
Complex.prototype.mandelbrot=function(){
var x0=this.x;
var y0=this.y;
var x=x0;
var y=y0;
var count;
var x_, y_;
var max_iteration=this.max_iteration;
function inSet(x,y){
return x*x+y*y<4;
}
count=0;
while(count<max_iteration&&inSet(x,y)){
x_=x*x-y*y+x0;
y_=2*x*y+y0;
count+=1;
x=x_;
y=y_;
}
return count;
};
```

当Worker正在进行Mandelbrot集合计算时，其主要事件循环被阻塞。所以UI进程不可能发送给它一个新的计算任务，或者更准确地说，Worker将不接收新的任务，直到完成当前任务。

要中断或改变Worker的行为，例如，要让用户在用户界面中选择绘制Mandelbrot集合的那个区域，然后要求Worker绘制该区域，有几种方法可以实现。

最简单的方法是终止该Worker并创建一个新的。这有一个优点，新的Worker以干净的状态开始，没有之前运行的Worker遗留下来的东

西。另一方面，这也意味着Worker必须从头开始加载所有的脚本和数据。因此，如果Worker的启动时间很长，这可能不是最好的办法。

第二种方法稍微复杂一些：通过程序手动管理任务队列。在主线程中或包含一个数据块列表的Worker中计算数据结构。当一个Worker需要任务时，它可以发送一个消息到该队列对象，让它发送一个任务。这种创建更复杂，但有几个好处。首先，在应用需要做不同的事情时，Worker并不需要重新启动。其次，允许使用多个Worker。当需要问题的下一部分时，每个Worker可以查询队列管理器。

也可以让主任务按顺序发送大量的事件给Worker。然而，这样做有一个问题，JavaScript没有办法清除事件队列。因此，使用这样一个可以管理的作业队列，似乎是最好的办法。在下一节，我们将探讨这一解决方案。

没有规定一个应用中只能用一个网络Web Worker。JavaScript很乐意让用户启动合理数量的Worker。当然，这只有当问题可以很容易地分割成几个Worker时才有意义，很多问题都可以做到这一点。

每个Worker是一个独立的构造，因此可以用同样的源代码创建多个Worker，或者创建多个独立工作的Worker。

Worker是JavaScript中合理的重大结构，因此为一个给定任务创建十个以上Worker可能是一个坏主意。然而，最佳数量可能要根据用户

的浏览器和硬件以及要执行的任务来确定。

测试和调试Web Worker

在过去十多年里，JavaScript调试工具已经变得相当不错。Firebug和Chrome开发者工具都是一流的调试工具，可以用来测试JavaScript应用。遗憾的是，它们不能访问Web Worker中运行的代码。因此只能设置断点或按步调试Worker中的代码。Worker也不能在列表中显示出现在Firebug和Chrome相应script标记中的已加载脚本。Selenium或QUnit测试也不能直接测试在Web Worker中运行的Worker代码。

Worker中的错误被报告给Firefox和Chrome的控制台。当然，在许多情况下，知道错误发生在哪一行和哪一个文件没有太大帮助，因为实际的错误可能在别处。

Chrome没有为程序员提供调试Web Worker的方法。Chrome开发者工具"script"面板包含一个"Web Workers"复选框。此选项会让Chrome用iframe模拟一个Worker。

多线程复用模式

可以使用Web Worker从用户浏览器任务中取出复杂的任务，为程序员提供强大的力量。Firefox自3.5版本开始支持Worker,Chrome从4.0版本开始支持Woker。同样，Safari和Opera也从某个时间开始支持Worker。然而，截至目前微软Internet Exporer还不支持Web Worker（虽然它可能会在IE10中出现），iOS上的Safari中也不支持Web Worker，因此iPad/iPod/iPhone平台不可能支持Worker。

理想的方法是用一个类库，使程序员能将要运行的代码抽象成一个函数或模块，用最佳的方式在后台自动运行这些代码，要么是当Web Worker可用时，要么通过setTimeout方法。此外，这个类库应该能够提供一套通用的接口，用于各种交互，如将一条消息发回主应用。

这样的类库应该用功能检测获取要运行代码的版本，而不是用浏览器检测。虽然现在有的浏览器支持Web Worker有的不支持，但是将来会改变这种状况，而且类库应该能够在这些变化出现时仍能工作。

此模式下实际工作的函数将以运行状态为参数重复调用，完成任何它需要做的处理，并返回一个修改后的状态参数，用于再次调用直到完成工作，然后调用stop()方法或以其他方式中断。run函数（见例8-6）应作为一个纯函数处理，它应该只处理其输入并返回一个值，而不

会对全局状态的变化有任何影响，根据它是否作为一个Worker运行，将有一套不同的接口集可用。

例8-6：Run

```
(function()
{
runner.setup(function(state)
{
this.postMessage({state: state});
return{
time: state.time+=1
};
}, {
time: 0
});
}());
```

当Worker运行时（见例8-7），run函数可以从一个标准循环的内部运行。该系统通过一些初始参数调用postMessage，然后将它们作为初始状态传递给run方法。该方法将通过while循环反复调用，直到它调用stop函数，状态将在这一点发回主消息。

例8-7：用Web Worker运行函数

```
var runner=
{
stopFlag: false,
postMessage: function(message)
{
self.postMessage(message);
},
stop: function()
{
this.stopFlag=true;
```

```
},
error: function(error)
{
this.stopFlag=true;
},
setup: function(run)
{
this.run=run;
var that=this;
self.onmessage=function message(event)
{
that.execute(JSON.parse(event.data));
};
},
execute: function(state)
{
var that=this;
setTimeout(function runIterator()
{
that.state=that.run.apply(that, [that.state]);
if(that.stopFlag)
{
that.postMessage(that.state);
}
else
{
that.execute();
}
}, 16);
}
};
(function()
{
runner.setup(function(state)
{
var newstate=state;
//modify newstate here
return newstate;
}, {
time: 0
});
}());
```

当浏览器的Web Worker不可用时，这个函数应该以短时间重复超时的方式运行，而不是放在while循环中（见例8-8）。while循环会阻塞消息队列，导致主线程不能发送消息到Worker。这个库的主要目的就是用超时来释放主线程，同时通过消息根据需要改变函数运行的状态。

与之前一样，runner要用回调函数返回的state作为参数再次调用run函数。但是，由于它不是一个Web Worker，因此随后要调用window.setTimeout()方法经过一段时间后再显示下一个迭代，并再次调用该函数。

例8-8：不用Web Worker运行函数

```
var runner=
{
stopFlag：false，
//override this function
onmessage：function(msg)
{
if(msg.state)
{
var state=msg.state；
$('#status').html("time："+state.time)；
}
if(msg.set)
{
this.state=msg.set；
}
return this.state；
}，
postMessage：function(message)
{
this.onmessage(message)；
}，
```

```
stop: function()
{
this.stopFlag=true;
},
error: function(error)
{
this.stopFlag=true;
},
setup: function(run,state)
{
this.run=run;
this.state=state;
this.execute();
},
execute: function()
{
var that=this;
setTimeout(function runIterator()
{
that.state=that.run.apply(that, [that.state]);
if(that.stopFlag)
{
that.postMessage(that.state);
}
else
{
that.execute();
}
}, 250);
}
};
```

模拟的Web Worker与代码主体之间的通信也有所不同。由于之后没有回调的postMessage方法，运行者必须通过提出一个注册回调机制来模拟，该回调可以采用与Web Worker的onmessage处理函数相同的参数。

本模型提出了一个创建在Web Worker和一般的JavaScript之间可移植代码的模型，这不是一个完整的解决方案。它缺少一些功能，如加

载代码。还缺少调用异步方法（如一个Ajax调用）的策略和完成时的恢复处理。这很有必要，通常Web Worker被设计用于处理器密集型的工作，有时在访问一个Ajax调用或IndexeDB时有意义。

Web Worker库

在主线程中进行JavaScript编程时，程序员用一个库，如jQuery来改善的API和隐藏浏览器之间的差异。对于使用Web Worker，有一个名为jQuery Hive（http://github.com/rwldrn/jquery-hive）的jQuery的扩展提供了此功能。Hive包括JavaScript主线程中的PollenJS库，该库包括创建Worker的接口。

如果需要的话，Hive还将编码和解码主线程和Worker之间的消息。在某些浏览器中（尤其是Firefox），复杂的数据可以通过postMessage接口发送。然而，在Chrome和Safari的一些版本中，postMessage的只处理字符串或其他简单的数据。

Hive还在Worker中包含jQuery API的一个子集。在Hive API中最重要的方法是$.get()和$.post()，对应jQuery中的响应API。例如，如果一个Worker需要通过Ajax访问服务器，使用Hive更容易实现。

Hive还包括通过$.storage访问永久存储接口。要设置一个值，使用$.storage（name，value）。如果设置为不用第二个value参数，调用$.storage（name）将返回现有的值。

Hive中还包括$.decode()和$.encode()，可用于对JSON消息进行编码或解码。

第9章　Web Socket

HTTP是一个请求和响应协议。其设计目的是请求文件，并围绕请求文件的思想进行操作。这对于必须先加载数据然后再进行保存的应用类型非常有效。

然而，对于需要服务器实时数据的应用效果相当糟糕。许多应用类型要求实时或半实时访问服务器。例如聊天或那些实时共享数据的应用，如许多Google Office应用，对服务器来说确实需要一个方法在服务器上发生某事时向浏览器推送数据。HTTP中有几种实现方式，但没有真正有效的方法。

一些应用，如Gmail，只简单地建立了一系列大型的HTTP请求序列，每秒一次以上（见图9-1）。这样产生大量的开销，而且也不是一个特别有效的服务器轮询方法。同时，还会产生大量的服务器负载，因为每个请求都需要在服务器上建立和销毁，以及HTTP头和用户身份验证的网络开销。HTTP头可以给每个请求增加几百KB。在一个繁忙的服务器上，这会给服务器和网络增加相当数量的负载。

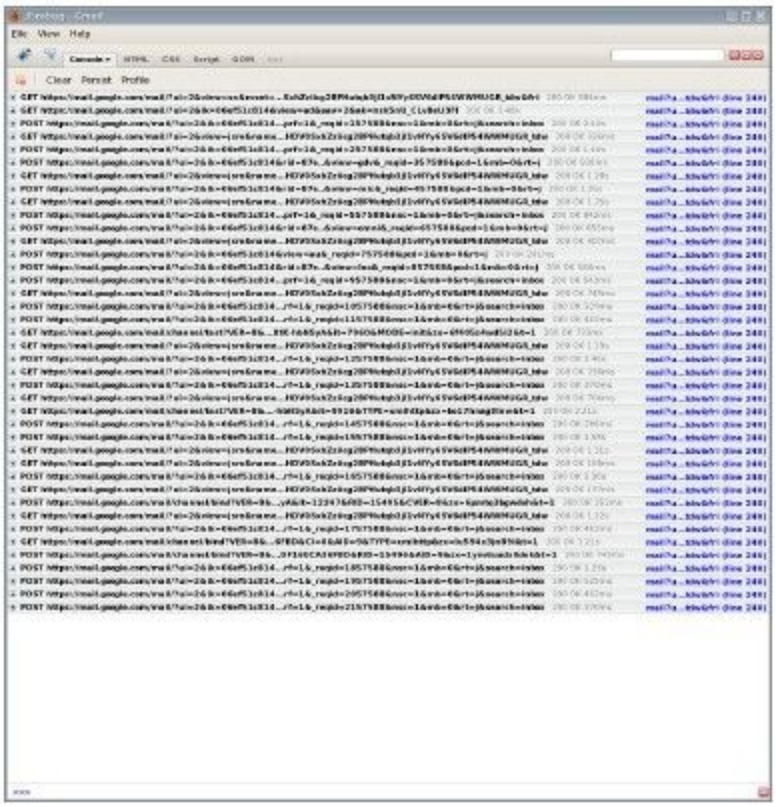

图　9-1　Firebug调试Gmail

第二种方法是打开一个到服务器的HTTP请求，并把它挂起，当服务器需要发送一些数据时，它会发送到客户端，然后关闭HTTP请求。此时浏览器会打开一个新的连接并重复这个过程。根据所采用的特定服务器技术，当持续运行一个大型的线程池和连接时，即使处于等待状态仍会在服务器上导致显著的负载，尽管在使用非阻塞服务器（如Node.JS）时，这几乎不成问题。更复杂的是，因为该浏览器可能只允许在给定时间向给定服务器发送有限数量的Ajax请求，因此为处理一个或两个请求打开它可能会导致别的工作被阻塞，这不是实现的最佳方法。

HTML 5引入了Web Socket的思想，与TCP/IP套接字的工作方式非常类似。浏览器向目标服务器打开一个套接字，并保持开放直到它不再需要或者显式关闭为止。套接字是一个双向实时数据通道，而一个HTTP请求是一个简单的轮询系统。如果用Ajax通过HTTP发送每个键点击到服务器，那么有可能产生至少300～400KB的开销，用cookie处理每次按键可能需要一两千KB。没有用于套接字的HTTP头，将减少很多开销，开销将减少到只有几个KB。

截止到本文编写时（2011年8月），WebSocket已经在Chrome8及其更高版本和Safari5版本中得到支持。Firefox版本6中包含了对Web Socket的支持，但构造器为MozWebSockets。Opera已经实现了Web Socket规范，但是由于安全问题有待解决，Web Socket默认为关闭状态，对于不支持Web Socket的浏览器可以使用传统的HTTP或者Flash进行回调。

有一些类库，如socket.io（http://socket.io），将为Web Socket提供一个持续的接口，并向可能不支持Web Socket的浏览器提供回调用于老式HTTP通信。也可以在支持Flash但不支持Web Socket的浏览器中用Flash模拟Web Socket。

Web Socket规范文档也似乎是一个进展中的工作。虽然Web Socket已部署在一些浏览器中，但是关于它们如何实现的文档还很

少。已经出现了一些Web Socket标准的早期版本，但是它们还互相不兼容。

Web Socket接口

要使用Web Socket，首先需要创建一个WebSocket对象，并传递一个Web Socket URL作为参数。不同于HTTP方法，Web Socket URL以ws或wss开头，一个安全的Web Socket将使用SSL.类似于Ajax下的HTTPS：

```
var socket=new WebSocket("ws：//example.com/socket");
```

一旦套接字的连接打开，将会调用套接字的socket.onopen()回调，让程序知道一切准备就绪。当套接字关闭时，socket.onclose()方法将被调用。如果浏览器希望关闭套接字，应该调用socket.close()。

用套接字发送数据使用socket.send（"data"）方法。数据限定为字符串，因此如果是比简单的字符串更复杂的数据，需要将该数据编码为JSON、XML或其他数据交换格式。此外，套接字仅处理文本，如果必须发送的数据是二进制数据，应该通过一些方法将其编码成文本。

建立Web Socket连接

Web Socket连接开始很像一个HTTP连接。它在端口80（WS）或443（WSS）上打开一个到服务器的连接，然而，除了标准的HTTP头，它还包括一些新的报头，告诉服务器这是一个Web Socket连接，而不是一个HTTP连接。它还包括一些用于提供安全的握手字节。Web Socket协议使用端口80和443，多数代理和防火墙需要进行正确处理。Web Socket也可以用与HTTP协议相同的方法来指定不同的端口，但是Web Socket调用（如一个Ajax）必须与产生它的Web服务器使用相同的端口。

一旦连接建立，连接的两端都可以用它发送数据。可以发送任何有效的UTF-8格式的字符串。具体的数据格式由服务器端和客户端共同决定。通常情况下，数据将可能是JSON或XML，如果需要也可以使用其他格式。

Web Socket示例

为了说明如何使用Web Socket，给出了例9-1中的简单示例。这里用一个非常简单的JavaScript函数向一个提供股票价格服务的服务器打开了一个套接字。它发送一个感兴趣的股票代码（IBM）。然后，服务器会找到该股票的价格并将其以一个JSON发回给客户端。服务器可以设置为每5s轮询一次新价格，并在价格变化时发送回客户端。客户端将只在每次价格变动时刷新元素。

例9-1：Socket客户端示例

```
$(function()
{
var socket=new WebSocket('ws：//localhost/stockprice')；
//wait for socket to open
socket.onopen=function()
{
socket.send(JSON.stringify(
{
ticker："ibm"
}))；
}；
socket.onmessage=function(msg)
{
var prices=$.parseJSON(msg.data)；
var html="IBM："+prices.ibm；
$('div.prices').html(html)；
}
})；
```

对任何用过Ajax的程序员来说，应该非常熟悉例9-1中处理Web Socket的浏览器代码。用相应的URL创建一个Web Socket对象。一旦Socket被打开（一定要等它打开），数据可以通过socket.send事件发送到服务器。当服务器将数据发送回浏览器时，通过事件对象的数据字段中的字符串调用socket.onmessage事件与事件对象的数据字段的字符串。在这种情况下，数据是JSON的，因此它可以用标准浏览器的JSON解析方法进行解析，然后显示在浏览器中。

如果没有服务器使用它，Web Socket客户端就没有多大意义。通常Web Socket用于处理事件驱动的数据，如共享文件、股票代码或聊天服务。虽然Web服务器开发往往选择PHP语言，但是对于本例，包含为长期运行程序和事件建立的编程模型的语言将更有意义。

这里有几个不错的选择。Node.js的效果很好，而且具有JavaScript（Web程序员已经熟悉了）的优势。其他可能包括：Erlang和Yaws，有一个Web Socket接口以及多处理器模式，可以作为此类编程的思路。还有一些选项用于Java和JVM的其他语言，包括Scala、Clojure等。也可以用Ruby以及可能大部分的.NET/CLR语言实现。说实话，大部分用于Web服务器编程的语言能够使用Web Socket。

在本例服务器端代码中（用Node.js实现），服务器是用websocket-server包建立的，可以通过NPM或者在github上找到。服务器在8080端口上等待连接，当收到一个连接时调用回调。该连接回调

通过连接对象等待消息到达。在本例中，随后调用一个高阶函数tickerUpdate，该函数查询股票价格并在对应股票价格改变时调用回调，发送新的价格给客户。Node编程更全面的指导查阅Tom Hughes-Croucher的著作《Node：构建与运行》（Node：Up and Running）。

例9-2：Socket服务器示例

```
var ws=require("websocket-server");
var server=ws.createServer();
server.addListener("connection", function(connection)
{
connection.addListener("message", function(msg)
{
var tickerSymbol=msg.ticker;
tickerUpdate(tickerSymbol,function(price)
{
var msg=
{
};
msg[tickerSymbol]=price;
server.send(connection.id,JSON.stringify(msg));
});
});
});
server.listen(8080);
```

Web Socket协议

虽然多数时候，Web Socket的底层细节与程序员无关，但是浏览器中和服务器上的接口将负责该细节，并提供一个可以发送数据的API。

话虽这么说，但是有时知道关于事情如何运作的底层细节，对于了解为什么某些东西不能正常工作或者在其他环境下实现Web Socket客户端可能是有用的。特别需要了解套接字是如何建立的。

Web Socket在浏览器和服务器之间使用TCP套接字而不是一个HTTP封套进行数据传送。但重要的是要了解如何建立一个套接字。当浏览器试图打开一个套接字时，它发送一个看起来像HTTP GET请求，但有一些额外的头的东西，如例9-3所示。

连接建立后，然后来回发送数据帧。每帧以一个空字节0x00开始，以一个0xFF字节结束。封套内部是UTF-8格式的数据。

例9-3：Socket头

```
GET/socket HTTP/1.1
Upgrade：WebSocket
Connection：Upgrade
Oragin：http://www.test.com
Host：www.test.com
Content-Length：0
```

有一些Web Socket的服务器端实现，可以在Python、Ruby、Erlang、Node.js、Java以及其他语言中工作。Web Socket的类库正在不断发展，有各种状态的开发包可用于Web开发用到的几乎所有主要语言中。一般情况下，Web Socket的服务器端将取决于项目的其他需要。因此，对于为给定项目使用的环境寻找Web Socket包很有意义。

Ruby Event Machine

Ruby Event Machine也为使用Web Worker提供了一个理想的平台，因为它向程序员提供了一个基于事件的接口，通过这个接口可以向客户端发送数据流。Event Machine：Web Socket接口与Java Script接口配合得很好。在客户端，EventMachine接口有用于onopen、onmessage和onclose的标准事件处理程序，也可以通过ws.send()函数把数据传回客户端。

例9-4显示了一个非常普通的Ruby Web Socket接口的"hello world"示例。

例9-4：Ruby EventMachine Web Socket处理程序

```
    require'em-websocket'
    EventMachine：WebSocket.start(：host=>"0.0.0.0"，：port=>
8080)do|ws|
    ws.onopen{ws.send"Hello Client!"}
    ws.onmessage{|msg|ws.send"Pong：#{msg}"}
    ws.onclose{puts"WebSocket closed"}
```

```
end
```

Erlang Yaws

Erlang是数十年前开发的非常纯正的函数式语言，当时主要用于电话的程控开关，后来发现在很多其他需要大量并行或者强健壮性的领域也很适用。它同时具有并行性、容错性和强拓展性。近年来它又用于网络世界，因为它所有对于电话程控开关方面的优化在网络服务器方面也非常有用。

Erlang Yaws网络服务器当然也支持Web Socket。相关文档和简单的代码示例可以在Yaws的Web Sockets页面（http://yaws.hyber.org/websockets.yaws）找到，如例9-5所示。

例9-5：Erlang Yaws Web Socket处理程序

```
    out(A)->
    case get_upgrade_header(A#arg.headers)of
    undefined->
    {content, "text/plain", "You're not a web sockets client!Go
away!"};
    "WebSocket"->
    WebSocketOwner=spawn(fun()->websocket_owner()end),
    {websocket,WebSocketOwner,passive}
    end.
    websocket_owner()->
    receive
    {ok,WebSocket}->
    %%This is how we read messages(plural!)from websockets on
passive mode
    case yaws_api: websocket_receive(WebSocket)of
    {error,closed}->
```

```
   io:format("The websocket got disconnected right from the
start."
   "This wasn't supposed to happen!~n");
   {ok,Messages}->
   case Messages of
   [<<"client-connected">>]->
   yaws_api:websocket_setopts(WebSocket,[{active,true}]),
   echo_server(WebSocket);
   Other->
   io:format("websocket_owner got:~p.Terminating~n",[Other])
   end
   end;
   _->ok
   end.
   echo_server(WebSocket)->
   receive
   {tcp,WebSocket,DataFrame}->
   Data=yaws_api:websocket_unframe_data(DataFrame),
   io:format("Got data from Websocket:~p~n",[Data]),
   yaws_api:websocket_send(WebSocket,Data),
   echo_server(WebSocket);
   {tcp_closed,WebSocket}->
   io:format("Websocket closed.Terminating echo_server...~n");
   Any->
   io:format("echo_server received msg:~p~n",[Any]),
   echo_server(WebSocket)
   end.
   get_upgrade_header(#headers{other=L})->
   lists:foldl(fun({http_header,_,K0,_,V},undefined)->
   K=case is_atom(K0)of
   true->
   atom_to_list(K0);
   false->
   K0
   end,
   case string:to_lower(K)of
   "upgrade"->
   V;
   _->
   undefined
   end;
   (_,Acc)->
   Acc
   end,undefined,L).
```

如果你不懂Erlang的语法的话，这段代码会很难懂。代码的关键点在于，客户端可以发送任意组合的请求（比如TCP连接、Web Socket连接或者是带数据的请求）并且服务器端能按照发送请求的不同把每条消息都正确处理。

第10章　新标记

除了大量处理数据的新接口外，HTML 5还引入了一些新的标记，可以用于在一个Web页面中提高应用开发人员开发优质应用的能力。

应用标记

任何应用程序中都有的一个共同任务：向用户反馈一个长时间运行任务的运行进度。让用户知道任务正在进行，没有冻结。可以使用一些＜div＞元素和自定义的CSS显示一个进度标尺，但HTML 5通过一个新＜progress＞标记规范了程序和外观。截至本书编写时，该标记已经在Firefox和Chrome中得到支持。它提供了两个属性使其易于向用户显示进度：value显示进度条的当前值，max显示进度条的最大值。

为了向旧版浏览器的用户显示一个进度指示，建议在进度条内的＜span＞元素里包含一些形式的文字。例10-1显示了一个完成进度为20%的进度条的代码。例6-4显示了JavaScript如何将属性作为事件进行更新，从而在程序中指示进度。进度条也可以像任何其他HTML元素一样用CSS设置样式。

例10-1：meter指示器

```
<!DOCTYPE html>
<html>
<head>
<title>Progress</title>
</head>
<body>
<progress value="20"max="100">
<span>running</span>
</progress>
</body>
</html>
```

＜progress＞元素可以用于显示一个正在运行的事件，＜meter＞元素可以用于显示一个静态值，如一个磁盘有多少可用空间或者一个筹款目标增加了多少钱。

＜meter＞标记可以带一些参数，包括min、max、low、high、optimum和value。所有这些都应设置为数字值。min和max值显示值范围的两端，而value属性显示当前值。High、low和optimum参数允许标记将范围分成子段。通过CSS，可以用不同的样式值将范围的不同部分设置成不同外观。例如10-2演示了＜meter＞标记的用法。

像＜progress＞标记一样，＜meter＞标记内应封装一个＜span＞元素，用于在不支持此标记的浏览器中显示该数据。目前，Chrome与Opera都支持这个标记。其他浏览器可能会逐渐跟上来。

例10-2：meter指示器

```
<!DOCTYPE html>
<html>
<head>
```

```
<title>Meter</title>
</head>
<body>
<meter min="0"value="20"max="100">
<span>20%</span>
</meter>
</body>
</html>
```

通过WAI-ARIA无障碍访问

使用Web应用（或任何其他的GUI应用程序）对于那些身体有残疾的人来说可能有很大的障碍。因此，HTML 5定义了无障碍富互联网应用（ARIA）。尤其针对视觉障碍（其中可能包括出于某种原因，仍在使用非图形浏览器的用户）。通过在应用的标记中添加属性，可以帮助这些用户访问标记中的内容。

这部分不是一个对无障碍访问应用的完整指南，介绍无障碍访问应用需要整本书。WAI-ARIA的基本思想是为页面上的元素增加可以通过屏幕阅读器读取的语义，使有视觉障碍的用户了解页面上发生了什么。＜img＞标记的alt属性和＜hr＞标记的title属性对这种文本的支持已经有一段时间了。

WAI-ARIA的最常见的属性是role属性。它为元素提供了上下文环境。在HTML中，如＜span＞和＜li＞之类的元素用于做许多不同的事情，从导航到实际列表。通过增加role属性，可以帮助屏幕阅读器识别所有这些内容。例10-3演示了WAI-ARIA的用法。

例10-3：WAI-ARIA

```
<!DOCTYPE html>
<html>
<head>
```

```
    <title>Meter</title>
    </head>
    <body>
    <ul id="tree1"
    role="tree"
    tabindex="0"
    aria-labelledby="label_1">
    <li role="treeitem"tabindex="-1"aria-expanded="true">Fruits
</li>
    <li role="group">
    <ul>
    <li role="treeitem"tabindex="-1">Oranges</li>
    <li role="treeitem"tabindex="-1">Pineapples</li>
    ...
    </ul>
    </li>
    </ul>
    </body>
    </html>
```

---

microdata

有时给一组HTML标记增加机器可读的数据是非常有用的。例如，可以在一个模板中用这个过程将数据编码成一个页面，之后可以通过JavaScript读取。为了以标准化的方式实现这种过程，HTML 5创建了microdata的概念，它可以添加到HTML 5的标记中。

传统的HTML标记提供应如何在屏幕上格式化显示的信息，而不是信息本身。一个程序可以看到一个＜li＞标记，并知道它是一个列表中的项目，但不会知道是什么样的列表。它是一个出售书籍或者出席某个事件的列表吗？通过添加microdata可以提供该数据的相关信息，供以后编程使用。

在一般情况下，与应用程序相比microdata标记可能更多地用于网页中，但是可以调用它作为插件在一个页面中运行或者以其他方式管理一个页面的应用程序来使用。可以想象，一个HTML 5应用作为公司网站管理面板运行，给销售经理提供了空间，可以加入microdata标记为出售项目添加相关信息，Google搜索引擎或者JavaScript程序可以用它形成外部供应商，该信息可能会被加入到一个网页中。

事实上，microdata非常简单：只是一些附加到HTML标记中的具有规范名称的额外属性，以及一些使用该数据属性的接口。

要指定页面中使用microdata的部分，需要给封闭元素添加一个itemscope。microdata的词汇库定义通过将itemtype属性设置为一个定义该词汇库的URL来实现。其中，itemprop属性指定封闭HTML标记的特定属性。

在http://www.data-vocabulary.org可以找到一些预定义的词汇库。包括用于Events、Organizations、People、Products、Reviews、Review-Aggregates、Breadcrumbs、Offers和Offer-Aggregates的规范。Google能理解这些语义，从而提高搜索结果。

在理论上应该在DOM中有接口解析microdata，但截至目前还没有准备好，而且它的实现也可能参差不齐。值得庆幸的是，microdata只是HTML属性，因此它可以用CSS选择器通过DOM接口或jQuery很容易地进行解析和处理。

新的表单类型

HTML 5用一大堆新的表单类型对自20世纪90年代初以来HTML中表单的经典形式进行了增强。其中大部分是关于从开始一直在使用的经典＜input type="text"＞标记的变化。这些新的表单类型对一个表单所能接收的输入类型和其所提供的接口提供了急需的灵活性。

许多情况下，在移动设备上改变输入类型也将导致设备生成一个自定义键盘，使用户能够输入正确的数据类型。例如，如果type设置为number，该设备可以生成一个数字键盘。对于tel类型，设备会生成一个看起来有点不同的数字键盘，不过为输入电话号码进行了优化。对于email类型，键盘将是一个为了输入电子邮件地址进行了修改的标准QWERTY键盘。

对智能手机应用特别有用的一种输入类型是语音输入类型：将itemtype属x-webkit-speech/＞。speech标记将接收用户所说的话并把它翻译成文字。例如，我的Android手机，有一个Google搜索小工具，可以通过语音搜索。speech标记也允许用户正常地键入文本。当用户说话时，input将触发的webkitspeechchange的事件，它可以用来与用户交互。

警告：对于非英语母语或者其他非标准口音英语的用户，使用这个标记可能会非常困难。许多我的以色列和俄罗斯的同事发现，这些输入不是非常有用。它也可能对英语以外的其他语言提供有限的支持。因此，如果应用的用户讲波兰语或希伯来语，标记可能不一定有用。

HTML 5增加了请求一个表单元素的能力。如果required的属性是set并且该元素是空白的，那么在CSS样式中它可能被设置为：invalid selector。

要显示一个滑块用于让用户从一个值的范围中进行选择，可以使用range类型。指定其最小值、最大值以及起始值。

其他输入类型相当简单，多数让程序员指定预期什么样的数据，并且如果一个字段不正确则让浏览器将其标记为无效。表10-1显示了一些可用的选项。

表10-1：表单输入

类型	用途	说明
email	Email地址	
date	日期	min和max可以指定一个范围
time	日期时间	min和max可以指定一个范围
tel	电话号码	通过正则表达式指定格式的模式
color	颜色	格式如 #BBBBBB
number	数字	将显示向上和向下箭头
search	搜索	min和max可以指定一个范围

audio和video

HTML 5还通过＜audio＞和＜video＞标记对音频和视频提供了新的支持。对于在HTML＜img＞标记中通过src属性使用过音频文件或视频文件的人应该对这些标记非常熟悉。它们都可以通过controls属性进行设置以显示控制。音频和视频也可以用JavaScript进行控制，用CSS设置样式。

对于如何编写HTML 5媒体脚本的完整描述超出了本书的范围，不过可以在ShellyPowers的书《HTML 5 Media》（HTML 5媒体）中找到相关内容。

Canvas和SVG

除了提供声音和视频支持外，HTML 5还对于在浏览器中用Canvas和SVG构建图像提供了支持。传统的HTML可以显示图片但是Canvas和SVG可以做更多的事情。

SVG是一种可伸缩矢量图形的XML标准，这就是说SVG中创建的图像可以缩放、旋转和操纵。此外，在SVG图像中的每个元素都是一个DOM元素。因此，可以说在SVG创建一个附加标准JavaScript事件处理程序的圆圈。SVG中的元素也可以作为DOM的部分进行操纵，元素可以直接用DOM或jQuery进行添加、删除或更改。SVG元素也可以像其他任何HTML元素一样用CSS设置样式。有的书深入探讨了SVG，如Kurt Cagle的《HTML 5 Graphics with SVG＆CSS3》（用SVG和CSS3绘制HTML 5图形）。

Canvas由苹果公司最初创建用于OS X中，后来引入到Safari中，已经从Safari引入到大多数其他浏览器中。Canvas提供了一个2D绘图表面，可以在上面用代码渲染图像。在本书第8章的“Web Worker碎形示例”中用它来绘制Mandelbrot集，其实它的力量很强。在Steve Fulton和Jeff Fulton的《HTML 5 Canvas》中有完整的介绍。

除了二维Canvas外，基于WebGL的三维Canvas已经开始在一些浏览器中实现了。相关示例和教程见HTML 5 Rocks的教程，网址：http://www.HTML 5rocks.com/en/tutorials/three/intro/。

地理位置

顾名思义万维网是覆盖全世界的，但是它也有很多事情是本地的。如果我正在寻找一个比萨饼店，我可能希望找一个在我所在位置附近的。地理位置让浏览器通过几种机制确定用户的位置。如果GPS可用（因为许多智能手机上都有），将使用它获得可能精确到几米的位置。如果没有GPS的接入，那么浏览器就可以尝试使用蜂窝基站或WiFi集线器的信息，这些方法可能不够精确，但往往也够用。如果我们的目标是找到当地的比萨饼店，知道到几个街区的位置可能就足够了。在大多数情况下，Geolocation API要求用户批准该请求。

要得到用户的位置，用两个回调作为参数调用getCurrentPosition()方法，一个回调用于返回成功的位置，另一个用于返回错误。在浏览器能够找到用户位置的情况下，将返回该位置的经度和纬度，如果能弄清楚海拔高度的话，海拔高度也是一个精确的参数。如果出现错误，将调用相应的错误情况。

```
   navigator.geolocation.getCurrentPosition(userLocationCallback,errorCallback);
```

新的CSS

除了新的JavaScript接口和新的HTML标记外，HTML 5还增加了一堆新的CSS选择器，其中包括：nth-child()和：first-child，以及CSS的“非”运算符：not()，其用法如：not（.box）。这些让开发者能更多地控制应用的界面。

除了新选择器外，HTML 5包含对CSS中的Web字体的支持。现在，可以在CSS中定义一种新的字体并在CSS中包含一个True Type字体文件用于产生特别的样式。

HTML 5的CSS还包括其他一些非常酷的功能，可以在文字的Overflow中作更多的设置，通过设置文字的Strokes设置一个DOM对象中透明度，通过HSL指定颜色以及更多的功能。当然，像HTML 5中的其他所有功能一样，不是所有的浏览器中都支持这些功能。

附录A　需要了解的JavaScript工具

在许多方面，JavaScript都是一种年轻的语言。虽然，它已经产生大约15年左右了，但是最近才被用于大型项目。因此一些帮助程序员编写强大、可调试的程序的工具仍在开发中。下面介绍几个强烈推荐使用的工具。

JSLint

是Douglas Crockford的JavaScript语法检查器。JSLint可以测试JavaScript代码所有可能出现的问题方式。可以从网站（http://jslint.com）或本地运行JSLint。在雅虎的控件集中有一个拖曳控件，可以从命令行使用Rhino（稍后讨论）运行它。一个简单的Bash脚本可以使它易于运行（见例A-1）。甚至可以把JSLint挂接到你的编辑器中。

如果你要使用一个JavaScript工具，那应该是这一个。要理解为什么，参见Crockford的《JavaScript：语言精粹》（JavaScript：The Good Parts,O'Reilly出版）。这本书详细描述了，为什么要选择JSLint。它能捕获很多非常常见的JavaScript错误，并且应该被认为是每一个JavaScript程序员的工具链中的一个重要组成部分。

要对每个文件配置JSLint，可以在该文件的顶部添加注释。这些注释可以在文件中开启或关闭选项，并告诉JSLint在其他地方定义的全局变量。

例A-1：JSLint shell wrapper

```
#!/bin/bash
java-jar～/bin/rhino/js.jar～/bin/jslint.js$1
```

JSMin

对JavaScript程序员来说，事实是程序会以源代码的形式下载到Web浏览器。可以通过缩小文件大小来提高速度。JSMin在文件上执行一些操作，例如删除注释和空白。通常，运行JSMin后文件大约是其原始大小的30%。这意味着要传输到客户端更少的字节。

警告：JSMin是一个单向的过程，因此一定要确保手头有一份文件的副本。也不应该用于开发中，因为它会使调试非常困难。并且确保运行JSLint之后再运行JSMin。

JSBeautifier

如果你的JavaScript代码常常变得混乱，JSBeautifier（http://jsbeautifier.org）是一个很好的工具。它会根据一些基本的规则重新格式化一个JavaScript文件。可以从网站或从桌面命令行使用

Rhino运行它。本书中的所有JavaScript代码已使用此工具进行了格式化。

JSBeautifier可以采用一些命令行选项来指定缩进样式和括号样式。可以通过制表符或空格缩进。-i选项控制缩进的级别。

JSBeautifier也可以格式化JSON字符串。因为它是用JavaScript编写的，所以可以把它嵌入到其他的JavaScript程序中。在某些时候，如果需要向用户显示一个JSON结构，可以使用这个类库漂亮地输出它（见例A-1）。

例A-2：JavaScript Pretty Printer

```
#!/bin/bash
cp$1$1.bak
export WORKING_DIR=pwd
cd～/bin/js-buautify
java-jar～/bin/rhino/js.jar beautify-cl.js-d～/bin/js-buautify/\
-i 1-b-p-n$WORKING_DIR/$1＞/tmp/$1
mv/tmp/$1$WORKING_DIR/$1
```

Emacs JS2模式

Emacs JS2（http://code.google.com/p/js2-mode/）模式是一个非常好的编辑JavaScript的框架。对于那些已经熟悉Emacs的人来说，这是非常有益的。

Aptana

对于那些希望在一个完整的IDE中开发JavaScript的人，Aptana是一个不错的选择。Aptana（http://www.aptana.com）是一个为JavaScript定制的Eclipse版本。有很多可以自定义的选项。

警告：Aptana将重新格式化代码，但它有时会以很奇怪的方式完成——很不错的方式，只是有点不同。

YSlow

在一个大型的We b应用中，加载缓慢是不正常的。如果发生这种情况，可以和Firebug一起使用这个工具，找出瓶颈在哪里。该工具配有一本由YSlow（http://developer.yahoo.com/yslow/）创建者撰写的书《高性能网站》（High Performance Web Sites）。它涵盖了远远比JavaScript更多的内容，它会在当每个文件作为网页的一部分传送时予以显示。

FireRainbow

FireRainbow（http://firerainbow.binaryage.com/）是一个Firebug插件，在Firebug脚本标记中设置JavaScript色彩。这是一个非常好的方式，使调试器中的代码更易于阅读。

Speed Tracer

Speed Tracer（http://code.google.com/webtoolkit/speedtracer/）是一个GoogleChrome插件，让你知道浏览器上正在干什么。在花费好几天时间优化JavaScript之前，知道它是否真正有益。如果CSS是真正的瓶颈，它会告诉你！

CoffeeScript

CoffeeScript（http://jashkenas.github.com/coffee.script/）是一种很酷的新语言，它使用JavaScript作为编译目标。如果你喜欢函数式编程，应该会对它感兴趣。它已经有一些忠实的追随者。有几本CoffeeScript书籍正在撰写或已经完成。CoffeeScript声称其产生的代码能够通过JSLint的验证。

ClojureScript

如果你喜欢Lisp，试一下ClojureScript（http://github.com/clojure/clojurescript），它是一种编译器，能将Lisp的Clojure直接编译成JavaScript。它是由Clojure的创造者Rich Hickey创建的。

Rhino

Rhino（http://www.mozilla.org/rhino/）是一种基于Java的JavaScript实现。如果你想创建在JavaScript中的命令行上运行的工具，Rhino可

以完成该任务。一些JavaScript程序，如JSLint，在Rhino下运行的与在浏览器中一样好。此外，Rhino可以用于在JavaScript中编写Java对象，这可能非常有用。

Node.js

Node.js（http://nodejs.org）是正在开发的一个新的服务器端平台。它使用JavaScript事件循环创建一个非阻塞的服务器，可以很高效地处理大量的请求。此项目未来将实现非常酷的功能。